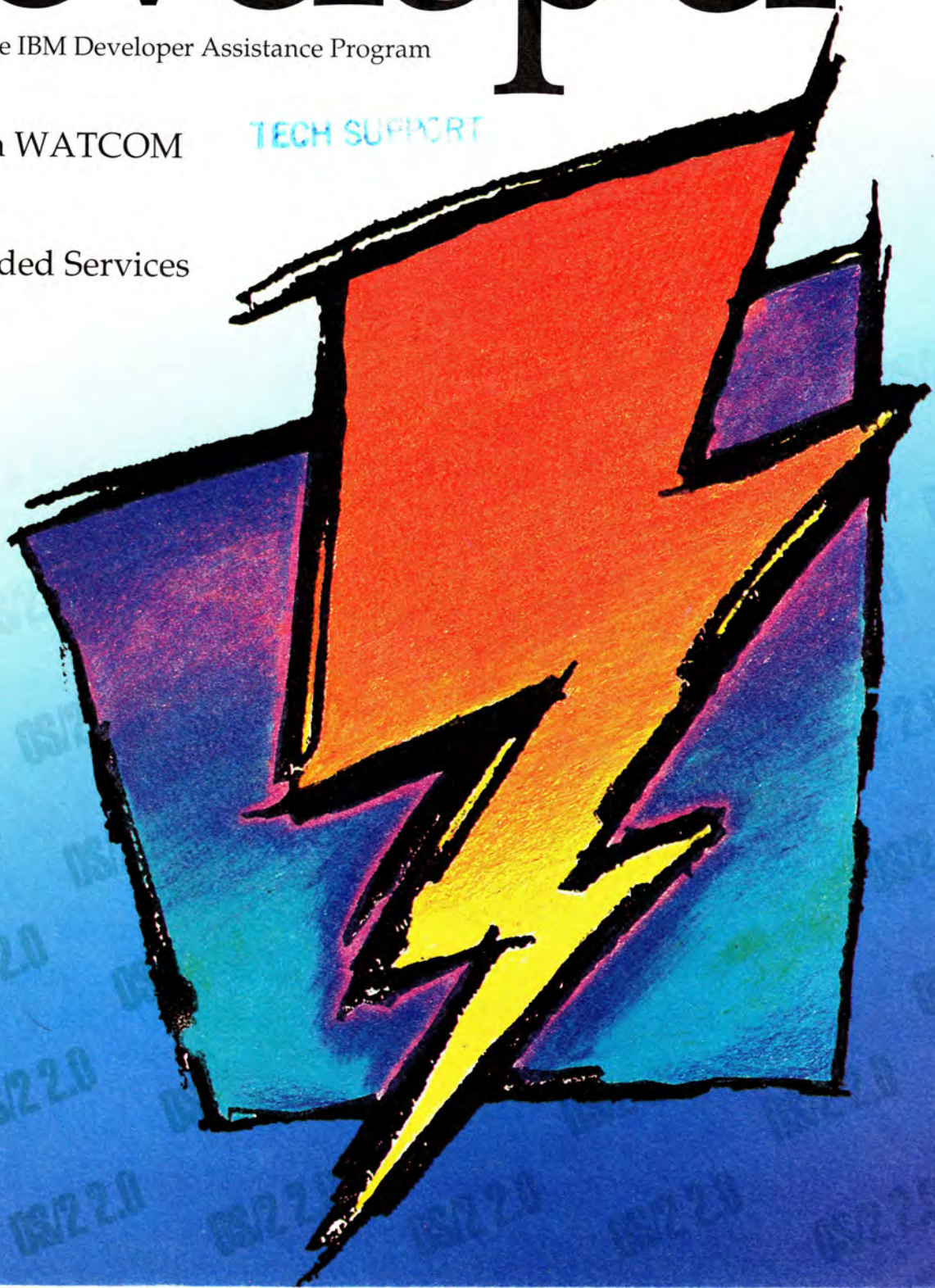
$9.95 USA

IBM Personal Systems Developer

A Publication of the IBM Developer Assistance Program
No. 2, 1992

- Spotlight on WATCOM
- 32-Bit Tools
- OS/2 Extended Services



IBM Personal Systems Developer

Table of Contents

Spring 1992, No. 2

The *IBM Personal Systems Developer* is published quarterly by IBM Software Support, Internal Zip 2230, 1000 NW 51st Street, Boca Raton, Florida 33429. Phone (407) 982-6408, FAX (407) 443-4233. IBM employees and branch office customers can subscribe to the *Personal Systems Developer* through IBM Mechanicsburg's Systems Library Subscription Services (SLSS) using the *Developer's* order number, G362-0001. Others can subscribe by calling the publisher, Graphics Plus Inc., directly at (800) READ-OS2. Subscriptions (U.S. only) are $39.95 yearly. International subscriptions are also available from the Publisher, phone (203) 366-4000, FAX (203) 368-9100. Questions, suggestions and article ideas should be sent to the Editor.

While back issues are not generally available, articles from the first seven issues of the *Personal Systems Developer* have been published in the *OS/2 Notebook: The Best of the IBM Personal Systems Developer*. This book can be bought at a local bookstore ($29.95) or by calling Microsoft Press at (800) MS-PRESS. Its Mechanicsburg order number is G362-0003-00.

Editor: Dick Conklin, IBM Software Developer Support. Publisher: Graphics Plus, Inc. 640 Knowlton Street, Bridgeport, CT 06608, Project Coordinator: Vickie Muratori.

IBM Personal Systems Developer is published by the Entry Systems Division of International Business Machines Corporation, Boca Raton, Florida, U.S.A., Dick Conklin, Editor.

IBM believes the statements contained herein are accurate as of the date of publication of this document. **However, IBM hereby disclaims all warranted either expressed or implied, including without limitation any implied warranty of merchantability or fitness for a particular purpose. In no event will IBM be liable to you for any damages, including any lost profits, lost savings or other incidental or consequential damage arising out of the use or inability to use any information provided through this publication even if IBM has been advised of the possibility of such damages, or for any claim by any other party.**

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you.

This publication may contain technical inaccuracies or typographical errors. Also, illustrations contained here may show prototype equipment. Your system configuration may differ slightly.

This publication may contain articles by non-IBM authors. These articles represent the views of their authors. IBM does not endorse any non-IBM products that may be mentioned. Questions should be directed to the authors.

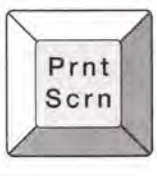
This information is not intended to be an assertion of future action. IBM expressly reserves the right to change or withdraw current products that may or may not have the same characteristics or codes listed in this publication. Should IBM modify its products in a way that may affect the information contained in this publication, IBM assumes no obligation whatever to inform any user of the modification. It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming or services that are not announced in your country. Such references or information must be construed to mean that IBM intends to announce such products, programming or services in your country.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever.

All specifications are subject to change without notice.

To correspond with the *IBM Personal Systems Developer*, please write to the Editor at IBM Corporation, Internal Zip 4607, P.O. Box 1328, Boca Raton, FL 33429-1328.

Trademarks



Editor's Comments

Dick Conklin

This issue marks our fourth year of publishing the IBM Personal Systems Developer. Thanks to all of you who have taken the time to send your accolades and suggestions by telephone, fax, Profs, BBS, and MCI Mail. Thanks also to the Florida Society for Technical Communication (STC) for awarding the Developer its Distinguished award for Periodicals in 1991. We are honored!

Also thanks to two special groups of readers: first, those who suggest new articles, and second, those who write them! Your message is clear: you prefer the technical, how-to, tips-and-techniques articles best. You are looking for the OS/2 features, functions, and software tools that not only simplify the development process but also help you produce quality applications. Our challenge in 1992 is to exploit the potential of OS/2 version 2.0. While we certainly need and appreciate our IBM authors, we like to hear from software vendors too. If you have a story to tell that others could learn from, let us hear from you.

This issue marks the return of our International column, written by IBM UK's Vicky Gallop. As OS/2 has increased in popularity around the world, so have our subscriptions and our contributors. Inside you'll find articles from Canada, the U.K., and Spain. Our Spotlight feature is on a Canadian company, WATCOM — a pioneer in language compilers.

Inside you'll find some articles on other topics you've asked for. What's IBM done for you lately? Did you know that we have a worldwide software manufacturing and distribution center that can replicate your diskettes, print your manuals, shrink wrap your package, and even ship to your customers? Meet Paul Delaney, manager of IBM's Worldwide OEM Product Software Manufacturing group.

Have you wanted to develop a leading-edge, cooperative processing application, but lacked the skills to do it? Don't despair; help is available. Read the article on SAA Express by Kevin Kornfeld. OS/2 Extended Edition has been replaced by OS/2 Extended Services. How has this changed Database Manager and Communication Manager support? Phil Sullivan and Mike Garrison have the answers. New and improved software tools continue to enhance the application development process. Philip Winestone of the IBM Toronto Lab reviews IBM tools for OS/2 version 2.0. Columnist Brian Proffit summarizes the range of 2.0 tool offerings. Other articles focus on Open+Build™, TalkThru™, Smalltalk/V PM®, SegMentor™, and FBSS™.

Our Spotlight articles always feature leading-edge software companies, and our next issue will be no exception. We're going to do something we haven't done before — introduce the OS/2 2.0 development team. If you'll pardon our pride, we are quite pleased with this new version of OS/2, and we thought it would be nice to meet some of the people who brought it to you and let them share a few war stories. Finally, congratulations to our project manager, Jo-Ann Campbell, who recently gave birth to a beautiful baby girl, Emily. This project was right on schedule — even a few days early!

Dick Conklin

Developer Assistance Program

Developer Assistance Program Update

Joe Carusillo

1992 OS/2 2.0 INTERNATIONAL TOOLS CONFERENCE

Our first OS/2 2.0 International Tools Conference, held in Fort Lauderdale last year, was a big success. The next one will be held in San Francisco, May 5 - 7 at the Hyatt Regency Hotel, San Francisco Airport. The conference is for software developers, software architects, and technical managers interested in learning more about the large number of development tools available for OS/2 2.0. It features a keynote address by Tommy Steele, the Director of the Personal Systems Programming Center in Boca Raton, demonstrations of 70 tools for OS/2 2.0, exhibits, a Help Room and over 40 different elective sessions. Session topics include:

- Language Compilers
- High Level Code Generators
- Object Oriented Environments
- Client / Server Tools
- General OS/2 2.0 Tools
- Multi Media Products

The $995.00 registration fee (after April 15) includes entry into all conference sessions and the exhibits floor, lunch each day, a copy of OS/2 2.0, and other mementos. One and two-day passes are also available. For more information and to enroll, call (800) 227-4374.

OS/2 DATABASE PERFORMANCE WORKSHOPS

The IBM Austin OS/2 Developer Support group is now offering OS/2 Database Performance Workshops. Each workshop is a combination of lectures and hands-on lab exercises based on the following topics:

- ES 1.0 Database Manager Overview
- OS/2 2.0 Systems Considerations
- Communications Configuration
- Distributed Database Connections
- Services Considerations
- Database Design
- Access Plans and the Optimizer
- Using the Explain Tool
- Database Programming Considerations
- Database Tuning Parameters

The workshops, held in Austin, Texas, provide in-depth technical instruction on performance-related aspects of the OS/2 Database Manager product. The four-day sessions are designed for developers who want to fine-tune their existing OS/2 database applications. Attendance is limited to 16 people per session.

The workshop fee of $1,500 per attendee includes technical instruction, lab assistance, use of IBM PS/2 server/requester systems, software tools and utilities. Breakfast and lunch are included; travel and other expenses are extra.

Workshops are planned for the weeks of May 5, July 21, September 15 and October 13, 1992. Workshop hours are 8:00 a.m. to 6:00 p.m. For enrollment information, call IBM Austin Developer Support at (512) 823-1588, Monday through Friday between 8:30 a.m. - 5:00 p.m., Central time.

OS/2 2.0 APPLICATION MIGRATION WORKSHOPS

The second quarter schedule for our Application Migration Workshops has just been released. Each Migration Workshop is an intensive 5-day, hands-on porting session where developers migrate their existing DOS, Windows 3.0, or OS/2 1.X product to a full 32-bit OS/2 2.0 product. They combine classroom lectures with extensive lab sessions, and are designed to allow a developer to complete a significant portion of an application port during the one week session.

2Q92 Schedule

- Windows 3.0 to OS/2 32-Bit PM 4/6/92
- OS/2 16-Bit PM to OS/2 32-Bit PM 4/27/92
- DOS to OS/2 32-Bit (VIO) 5/4/92
- OS/2 32-Bit (VIO) to 32-Bit PM 5/18/92

The registration fee for each workshop is $2,300 per person. This includes technical instruction, lab assistance, a Migration Workbook, meals and a license to an OS/2 2.0 Developer's Kit. Travel and Hotel accommodations are extra. For more information, call (407) 982-6408.

OS/2 2.0 EARLY CODE PROGRAMS

Now that OS/2 2.0 has become a generally available product, the OS/2 early code programs have been temporarily suspended until the first beta version of the next OS/2 release becomes available. Of course, technical support is still available through IBMLink for developers with ongoing projects. This having been the first large scale distribution of beta code by IBM Personal Systems, we have had our ups and downs over the last nine months with these programs. We appreciate all of the suggestions, comments, and criticisms that have come in along the way. Next time around, look for significant improvements like electronic download capability for developers as well as customers, and a CD ROM distribution option.

OTHER OS/2 2.0 EDUCATION

The IBM Atlanta Customer Education Center offers a range of courses on OS/2 for application developers and end users, including:

- Using IBM Developer's WorkFrame/2
- Using ENFIN/2 to Develop Object-Oriented Applications
- Using OS/2 Extended Services Relational Databases
- Migrating OS/2 1.X 16-Bit PM Applications to OS/2 2.0 32-Bit PM
- Installing and Using OS/2 2.0

These classroom sessions are offered at various IBM education centers. They can also be taught at a customer's site on request. For more information, call (800) IBM-TEACH.

HELP FOR OS/2 BOOK AUTHORS

Are you writing a book about OS/2 2.0? We can help you with:

- Enrollment in IBM's Developer Assistance Program
- OS/2 2.0 and Tools
- Publicity in IBM publications, seminars and trade shows
- Purchase and distribution of book by IBM

For more information, call Dick Conklin at (407) 982-1105, fax (407) 443-4233.

Joe Carusillo, *IBM Personal Systems, 1000 NW 51st Street, Boca Raton, FL 33431. Mr. Carusillo is the Manager of Software Developer Programs which includes the Developer Assistance Program. He started with IBM Entry Systems in 1982 and has worked on PC software since that time. He has a BS in Computer Systems Engineering from the Columbia University, School of Engineering and Applied Science, in the City of New York.*

Spotlight on WATCOM

WATCOM Compilers and OS/2 2.0: The Perfect Fit for Cross-Development

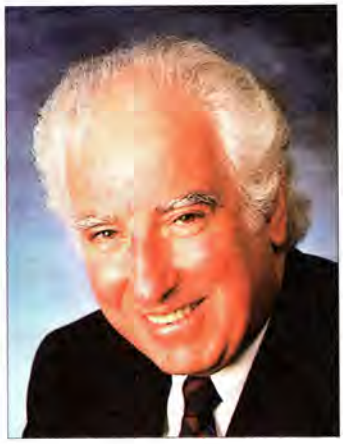
by Val Enright

The WATCOM Group, located in the university town of Waterloo, Ontario, not far from Toronto, develops and markets optimizing C and FORTRAN compilers and SQL database tools for DOS, Windows and OS/2™. Building on a solid reputation and experience in the higher education software market, the company has emerged as an industry leader in reliable optimizing compiler technology for the PC market.

Awards received by WATCOM C include: "Editor's Choice," PC Magazine (September 1988), "Best in Its Class," InfoWorld (May 1989), and "Jolt Product Excellence," Computer Language Magazine (February 1991).

UNIVERSITY ROOTS

During the middle '60s and early '70s, there were few tools that made effective use of computers for hands-on student programming. The Computer Systems Group, a small research and development group at the University of Waterloo changed that. This group created a set of tools that increased student access to the computer and provided help when students made errors. The tools were fast and easy to use, and the help for syntax errors gave professors more time to devote to teaching abstract concepts. Funding for the work done by the Computer Systems Group was derived from the success of WATFOR, the university's innovative FORTRAN compiler, which was developed for the IBM 7040 in 1965 and for the IBM 360 in 1967. This compiler featured run-time error checking in addition to syntax checking at compile time.



Wes Graham

The guiding force behind computing research and development efforts at Waterloo in those days was Professor Wes Graham, who joined the University after five years at IBM, and is now Chairman and Chief Executive Officer of WATCOM. Under Graham's directorship of the Computing Centre (1962 to 1973), the WATFOR™, WATFIV™, and WATBOL™ systems were developed. In 1973, Graham formed the Computer Systems Group to spark continuing research and development of educational systems.

The Group didn't do marketing back then according to Ian McPhee, President and Chief Operating Officer of WATCOM. "We belonged to a small club of people who used computers on campuses. Because we had software that solved some problems for

student computing, everybody wanted it. The small community grew and became a world-wide market. Eventually, we had 3,000 customers in over 60 countries."

Dave Boswell, WATCOM Vice President, Marketing and Sales, added, "Members of this small club often were members of SHARE (Society to Help Alleviate Redundant Effort), an association of IBM mainframe engineering and scientific customers. Professors who were teaching at Waterloo and working on the development of WATFOR would go to a SHARE conference and present papers on some aspect of teaching FORTRAN in the higher education environment. And, of course, the room would be packed with people facing the same challenges, who were eager to get their hands on the information."

Almost all the employees of WATCOM graduated from the University of Waterloo, including McPhee. "I did both my undergraduate and graduate studies there," He stated. "In the early '70s after I graduated, I began working with the Computer Systems Group. The group gained quite a bit of experience porting software. As each new computer system came out, we would reimplement. We realized we had to invest some of our development effort in working smarter, not harder. So we figured out ways to work portably. By the end of the '70s, we had a reasonable set of expertise in this area."

Ian McPhee

By this time, the University found itself in the somewhat awkward position of running a business without meaning to do so. This did not fit in with the University's mandate of research and teaching. "They were happy to spin off the business to us," explained McPhee. "We formed a company that is a subsidiary of WATCOM, which we finance and in which the University owns a small share. The company markets products, primarily to the education market, which are developed at the University or by WATCOM itself. An annual royalty stream of $1 million is paid to the University for its products.

Forming the company was a win-win situation for WATCOM and the University. For WATCOM, it meant instant access to a viable market and an opportunity to build on the already excellent reputation of the University. For the University, it meant getting rid of the headaches of customer support, distribution, the collecting of funds and other details associated with the business of producing software.

WATCOM was founded in 1981. In the company's early days, staff members made solid use of the expertise in portability technology they acquired from working with the Computer Systems Group. They designed and developed student-oriented language processors for APL, BASIC, COBOL, FORTRAN, and Pascal and leveraged these processors across the mainframe, mini and microprocessor range. The products became widely used; first in higher education and then–as the PC market exploded–in the K-12 market as well. Eventually, WATCOM was able to create a single product line for use in schools that supported a wide range of computing environments.

FIRST PC MARKET ENTRY

The requirement to support a software set that was portable across a broad range of machines was responsible for WATCOM's first entry into the mainstream PC market: a professional optimizing C compiler designed specifically for the PC software industry. McPhee stated, "We had a set of tools and compilers that we created for our use on the mainframe, midrange and micro systems we were supporting at the time. As we did more work on the PC and used the tools that were currently available, we realized they were significantly inferior in optimization quality to our internal tools. So we decided, there was a product opportunity."

When WATCOM launched its 16-bit optimizing C compiler in February of 1988, it won industry recognition because of its optimization capabilities. In addition to producing tight, fast code, the compiler was the first on the market to be entirely compliant with the new ANSI (American National Standards Institute) standard set for C. Boswell, an alumnus of Waterloo and the Computer Systems Group, remembered the event: "I joined WATCOM in 1987 during the final planning stages of marketing the compiler. Other software companies were claiming their compilers had the best optimizing technology. But it was our compiler that slipped in quietly and won most of the awards."

A FORTRAN FOLLOW-ON

Geno Coschi, OS/2 Tools Architect at WATCOM and also an alumnus of Waterloo and C. S. G. was a principle developer of WATFOR-77™, a portable successor to WATFOR and WATFIV, developed to support current ANSI FORTRAN across PC, midrange and mainframe systems. Coschi recalls his involvement with the design and development of WATCOM's optimizing FORTRAN compiler for the PC market. "In 1990 we released our 32-bit optimizing FORTRAN compiler which conformed to the ANSI language standard set in 1977. Its predecessor, WATFOR-77, was essentially a debugging compiler which emphasized good error diagnostics and quick turnaround in the development cycle. So we took this language support and molded it together with the optimizing code generator technology and supporting tools from the C compiler to make the WATCOM FORTRAN compiler."

Geno Coschi

WATCOM 32-BIT COMPILERS

"Each release of our 32-bit development tools has answered key customer requirements," McPhee said. "We shipped the industry's first 32-bit ANSI standard C compiler for DOS in 1989. Then in 1990 we delivered the first source-level debugger for 32-bit extended DOS as well as source-compatibility with Microsoft C. In August 1991 we bundled a 32-bit DOS extender and 32-bit Windows™ support, both with no run-time royalties. Previous to this, run-time royalty requirements on the 32-bit extenders were a key restraining factor in the move to 32-bit DOS programming. Now, in 1992, we're introducing support for OS/2 2.0™, as well as GO Corp's PenPoint™ operating system."

FORTRAN AND C INTEROPERABILITY

Because the FORTRAN and C compilers are based on common technology, object compatibility between the two is obtained in a straightforward manner. It's easy to write a FORTRAN program that calls C functions and vice versa. McPhee explained, "The run-time support for memory management and I/O is

based on the same low-level system interface. In fact, the FORTRAN logical output unit is the same as the C standard output file. So they don't conflict; instead, they work harmoniously."

Development tools, the debugger for example, are identical for FORTRAN and C. The debugger included with the purchase of the C package is also executable with FORTRAN.

"The debugger detects the language the module is written in and adapts to that syntax," added Coschi.

Boswell elaborated by saying, "The debugger is really a remarkable piece of technology. It's a single executable file that can debug programs running on the same machine or remote ones, 16-bit programs or 32-bit programs, FORTRAN programs or C programs. And it doesn't care whether the remote machine is hooked up by a Novell

Dave Boswell

LAN or an RS232 wire or a parallel cable. In fact, the program to be debugged could be on the same machine but in a separate DESQview™ session. The benefit is that developers have to acquaint themselves with only one debugger and can use it for all their debugging needs."

The WATCOM tool set also includes the Profiler, a performance-tuning instrument. By running a program through the Profiler, one gets a histogram that charts the execution times for each module of the program. You can then identify routines that use up too much execution time and fix the code so it's more efficient.

OS/2: THE 32-BIT PLATFORM CHOICE

As Boswell lists the design points of WATCOM products, they begin to sound familiar: compatibility, productivity, multiple targets. "What we see for the cross-development environment," Boswell stated, "is a confluence in design points between WATCOM compilers and OS/2 2.0. And, of course, the customer is the beneficiary."

He went on to say that, "DOS and Windows are features on the industry landscape. OS/2 recognizes this and provides execution environments for these products. Our multi-platform support meshes with these OS/2 2.0 capabilities. Using our tools and OS/2 as the host operating system, developers can compile and debug 32-bit applications for multiple target environments: DOS, Windows, and of course OS/2 itself."

Also available are FORTRAN and C compilers that generate 16-bit code. McPhee stated "As I mentioned earlier, our mission is to provide the best quality optimizing compilers for 16 and 32-bit processors for both C and FORTRAN, and to support the widest range of PC operating environments as targets for applications that are developed with these tools. Now we have a real operating system platform on which to do this work.

"As I'm sure you know, when you are developing new code, it has flaws and bugs. This is particularly true of a fairly low-level language like C that directly addresses memory with pointers. Frequently, new code does very bad things because it addresses memory it's not supposed to and hammers it. In unprotected environments like DOS and Windows, you might be corrupting part of the operating system. Even worse, it could be the part of the operating system that manages the files on disk. Who knows what nasty things might ensue? So it is much more comfortable to debug your applications, which are fresh and raw and have problems to be worked through, in OS/2 where you have a protected environment.

"When you do nasty things to OS/2, it bounces right back"
—Dave Boswell

"OS/2 supports productivity. With DOS or Windows, when you hammer the system because of a bug in an application you are developing, you must reboot the system. Sometimes you might have to power it off and on again. In some cases, it can take several minutes to get the system back up, and that really slows down the development process.

With OS/2, when you do those nasty things to it, it just bounces right back. You don't even have to reboot, and your productivity as a developer is unimpeded."

"Intel decided to enlighten us because we were doing so well in the dark"
—Ian McPhee

FULL SUPPORT FOR PM AND SYSTEM SERVICES

Both the FORTRAN and C compilers provide full access to PM function and systems services. Of course, full support might be expected for C. What comes as a surprise is that the FORTRAN compiler provides the same level of access to the system and PM as C. This means a developer can create a PM application using FORTRAN exclusively.

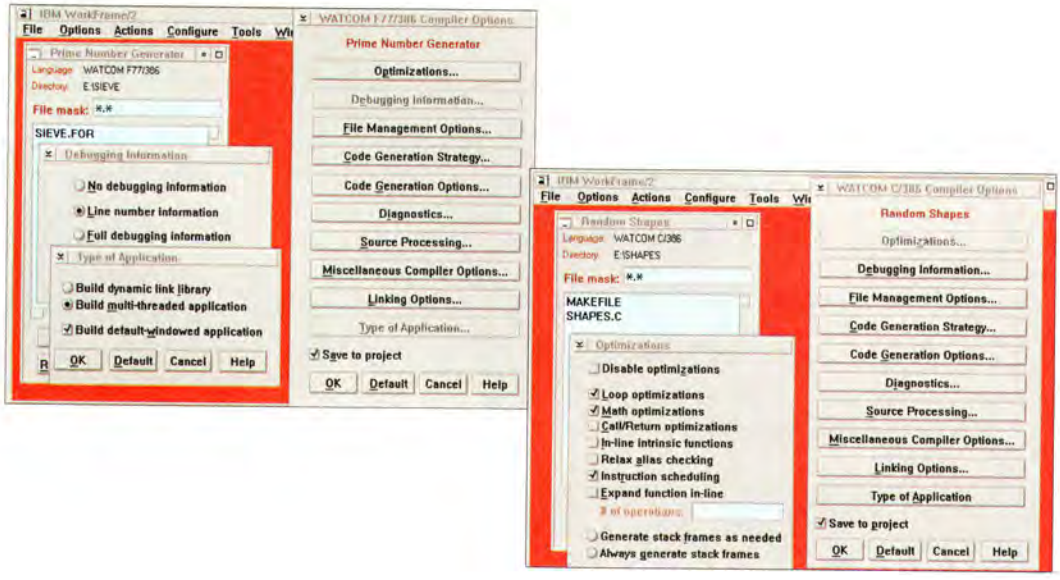
In the past, moving a FORTRAN character-mode program that ran on a midrange system to the PC presented a real challenge because the program had to shoe-horn into a 16-bit environment. OS/2 takes a big step toward enabling this migration by providing the 32-bit environment. According to Boswell, "The question is, how do you transform a character-mode application into a Presentation Manager™ application? Geno has created a default windowing library, which reduces the amount of work to zero. You link your FORTRAN application with the default windowing library, and it creates a PM application that looks like a character-mode session within a PM window. Of course the PM application doesn't have pull-down windows and so on, but your FORTRAN application is there, running under PM, and you didn't have to write any code to make it happen."

COMPATIBILITY WITH C SET/2

The WATCOM compilers "snap in" as components under the WorkFrame/2™ environment. In fall, WATCOM demonstrated this capability at Comdex and also at the International OS/2 Tools Conference. IBM also demonstrated its C Set/2 product under WorkFrame. WATCOM's integration with WorkFrame is important to IBM. "Our C compiler complements the C Set/2 compiler, which principally focuses on the development of OS/2 applications," stated McPhee.

PORTING FROM UNIX TO OS/2

In the OS/2 2.0 32-bit flat-memory environment, WATCOM C and FORTRAN



compilers enable porting of UNIX and Macintosh™ code easily. "That is why we are providing appropriate language tools to empower the folks that need to migrate from the UNIX platform to OS/2 2.0.," Boswell explained. "The ease of porting to OS/2 from UNIX is in direct contrast to the problems encountered when porting to the traditional 16-bit DOS environment. First, you must deal with the 640K barrier. Then you must account for the segmented architecture by using different classes of pointers. All these concerns are outside the architecture of the program's original 32-bit environment. Similarly, 16-bit Windows presents memory-management complications for programs being ported from 32-bit platforms."

INDUSTRY RELATIONSHIPS

For the past few years, WATCOM has worked very hard on industry relationships such as the one it has with IBM for OS/2 2.0 tools. WATCOM believes partnerships in the industry are invaluable for achieving credibility in the marketplace.

McPhee cites Novell as WATCOM's first 32-bit partner: "Our relationship with Novell helped launch us in the network arena. In 1989, Novell was about to release the 386 version of the network, and they wanted the best 32-bit C compiler they could find to support development of Netware™ applications. Netware has its own operating system that runs on the server. What they wanted was a compiler that was fully ANSI compatible and also state-of-the-art in its optimization techniques. The compiler would permit development and debugging of NLMs (Netware loadable modules) from a DOS or OS/2 1.X environment. So from a workstation you could develop a program, run it on a Netware server, and then debug it across the network. They came to see us because they read about our 16-bit optimizing compiler and saw how superior it was in terms of code size and speed."

Another key partner for WATCOM is Lotus™, which has been developing new products with both the WATCOM 16-bit and 32-bit compilers. Some Lotus applications have been converted from Microsoft's compiler to WATCOM's 16-bit compiler. Just by recompiling, they improved performance and shrank the size of their applications. "They are extremely pleased with the results," said McPhee. "Lotus has given us a great deal of feedback over the years on ways to improve our products and has identified key requirements of the software industry. We are very grateful to them for that."

McPhee continued by stating: "Our latest partner in strategic technology is Intel™. They recently shared designs of their new generation of processors and are collaborating on optimization techniques to get the most out of the chip's superscalar technology."

Intel is working with approximately 20 companies selected from the industry. Six are compiler vendors, which include WATCOM, Microsoft, MicroFocus and companies serving the UNIX market. "We think Intel decided to enlighten us because we have been doing so well in the dark," McPhee observed. "Maybe they also are aware that smaller companies like us can move faster to exploit the 32-bit technology."

WATCOM plans to have FORTRAN and C compilers which exploit Intel's new "P5" chip available before the silicon is cast for the chip. This will enable companies to have applications that exploit the P5 available at the P5 launch. "What we, WATCOM and Intel, are doing is shortening the cycle," McPhee explained. "Interested companies can get the tools from us to write applications that get maximum performance from the chip. Then, as soon as the silicon is ready to be mass-produced, their software will be there to support it."

McPhee stressed the importance of industry relationships; "We view industry relationships such as the ones we have with IBM, Intel, Novell, and Lotus as key to the success of WATCOM and the compiler tools business.

We have the P5 technology relationship with Intel combined with the 32-bit platform choice, OS/2 technology, and marketing relationships

with IBM. We believe these relationships will work well and that we will all win because of them. "

OTHER CANDIDATES FOR THE OS/2 DEVELOPMENT PLATFORM

Boswell described other WATCOM 32-bit target environments that are candidates for development on the OS/2 platform: "We mentioned a special class of programs that run under Netware. Companies such as Oracle and Gupta have announced that they have created versions of their relational database engines that run on the Netware server. These products are examples of Novell NLMs developed using our 32-bit compiler.

"PenPoint is an interesting system. The Pen Point prototype was released over a year ago as a 16-bit system. The real product is going out as a 32-bit system, and it was developed in WATCOM C/386. In fact, WATCOM C/386 is offered with GO's SDK for PenPoint. So, all the applications being developed by major companies to run on these pen-based systems are being developed with our 32-bit C compiler.

"Our compiler also creates code that is appropriate for embedded systems. (The term 'embedded systems' refers to computer systems used within other products, like the microprocessors which control VCRs and car engines.) We have the appropriate support so that our complier can be used with certain popular embedded systems development tools.

"One key vertical niche markets for 32-bit development right now is AutoCAD™ add-ons (called ADS or ADI products). Autodesk highly recommends the WATCOM compiler to their ADS development community. We have done a lot of work to ensure that we support them correctly and that we get to know the ADS customer. We are the only vendor with both C and FORTRAN tools for ADS development."

Boswell concluded by stating, "As you can see, our tools under OS/2 2.0 support a broad range of 32-bit target environments."

DOUBLE BYTE SUPPORT

The WATCOM compilers support use of double-byte characters used for Asian languages. At present, distributors are selling English compilers that support development of Kanji applications. However, work is being done to create Japanese versions of the product, which have messages, help text, and documentation in Japanese.

Last fall 1991, WATCOM's Japanese distributor, Lifeboat Japan ran an advertisement in major magazines in Japan that was a variation of the WATCOM lightning bolt ad. They took the WATCOM lightning bolt and superimposed it on the Tokyo skyline.

WATCOM TECHNICAL SUPPORT

When asked about WATCOM's technical support, McPhee had this to say: "Technical support is a key aspect of our business. We put strong emphasis on both the quality of our tools and the technical support for those tools. Because it is very hard to be perfect when you are writing an optimizing compiler, we must have a means of delivering prompt fixes to developers relying on our tools for critical applications. Also, we learn about the evolving needs of the industry by listening to our customers and placing a high priority on that in our company.

Deanne Farrar

"We have developed one of the best technical support groups in the industry. The quality of our technical support has contributed to the reputation we have established over the past three years."

Deanne Farrar, manager of WATCOM's technical support department,

describes the availability of technical support: "People can contact us by phone, fax, mail, or bulletin board. We do our best to keep everyone up and running. If we can't answer a problem right away, we try to get a work-around solution within 24 hours."

THE BULLETIN BOARD

WATCOM's bulletin board is by far the most popular means of contact for technical support inquiries from their customers. Callers can leave text messages and then resume their work. While they are working, their questions are researched, and the answers are often posted within 24 hours.

In addition to text files, customers can also upload binary files to the bulletin board. Farrar explained, "If the customer can send me a portion of the program that demonstrates a problem with the debugger or the compiler, this gives me an actual example to try. I look through the source code and compile it. Many times, it's just a programming error. We make sure they are using the right switches. The problem could be that they have just changed compilers and are not familiar with our option switches."

The bulletin board is also used for making patches (temporary code fixes) available. Rather than rebuilding the compiler each time a problem is discovered, a fix is posted on the bulletin board. Customers can download the fix and apply it to their compilers, using the Patch Facility that is packaged with the compiler.

WATCOM's OS/2 2.0 32-bit Languages Team
Seated l-r: Fred Crigger, Anthony Scian
Standing l-r: Geno Coschi, John Dahms, Jack Schueler, Craig Eisler and Brian Stecher

"In the case where a customer's deadline could be affected, we try to make a special patch available," said Farrar. "The other day I had a customer tell me, 'If I don't have a fix by Wednesday, it's going to delay my ship date.' I asked the customer to send us an example of the problem. After the file arrived on the bulletin board, I was able to determine that the customer had encountered a problem in the compiler. I went straight to our development people and told them, 'Look, if we don't get this fixed right away, we are going to cause the delay of this customer's ship date. If we can get a fix posted on the bulletin board by this afternoon, they'll be really happy.'"

Of course the customer wound up happy, as the fix was made and posted, and their deadline was met with time to spare. Farrar attributes this sort of result to a common sense of mission, "I have to credit our development organization for understanding that the business is centered around the customer."

COMPANY INFORMATION

WATCOM
415 Phillip Street
Waterloo Ontario
Canada N2L 3X2
1-800-265-4555 (in North America)
Tel: 519-886-3700 • Fax: 519-747-4971

Val Enright, *IBM Entry Systems Division, Boca Raton Programming Center, 1000 NW 51st Street, Boca Raton, FL 33429. Ms. Enright is a staff information developer and has been with IBM for twelve years. Her responsibilities include technical writing of OS/2 multimedia Presentation Manager/2™ programming information.*

Vendor Support

IBM Announces a New Service for Software Vendors

by Greg Slopey

IBM recently announced worldwide software manufacturing and distribution services for independent software vendors. We discussed this new venture with Paul Delaney, manager of Worldwide OEM Product Software Manufacturing Marketing.

DEVELOPER: What is IBM's goal in offering software OEM services?

DELANEY: We intend to become the premier worldwide vendor of replication, packaging, distribution and custom services for independent software vendors. Our emphasis is on total customer satisfaction, price competitiveness, project management and value-added services. By allowing us to perform these services, independent software vendors can devote their efforts to their primary task — writing and developing code.

DEVELOPER: What specific services will you provide?

DELANEY: We provide the same software manufacturing and distribution services to the outside world that we have given our internal IBM customers for many years. That includes replicating software and publications, customized packaging, as well as delivering products around the world. In addition, customers can receive package deliveries that maintain their idenity. In other words, their company name and logo will appear on the products and it will be transparent to the end user that IBM performed the manufacturing and distribution services.

DEVELOPER: Will customers be required to sign up for a total package deal?

DELANEY: No; customers may choose the full range of services or they can select specific options, such as only publications, media replication or translation services.

DEVELOPER: What expertise does IBM Software Manufacturing and Delivery (ISMD) offer its customers?

DELANEY: We are the largest software manufacturer and distributor in the world. Our facilities extend across five continents. In 1990, we managed the replication, packaging and distribution of approximately $10 billion worth of software, and handled 8,500 software products running on computers ranging from PCs to mid-range systems to mainframes. We shipped 8.8 million software packages containing supporting documentation in more than 30 languages. In short, no one can match our expertise in this area.

DEVELOPER: How does this benefit smaller companies?

DELANEY: It's a matter of leverage. Because of the high volume of manufacturing and distribution underway at ISMD, small companies can expect considerable savings in overall cost.

DEVELOPER: In addition to competitive cost advantages, what other benefits do you offer?

DELANEY: I believe our customers will benefit in several ways. First, they can realize savings from streamlining their purchasing activity. ISMD can be a single supplier for all the services a company needs, eliminating the need to manage many different vendors.

Paul Delaney

Second, when customers put their requirements in our hands, they can relax. We ensure quality at every step and offer one-stop shopping for software manufacturing and distribution. Third, we constantly test new technologies. That means our customers are assured of products at the cutting edge. Finally, we offer a full range of customized packaging options.

DEVELOPER: What media is used for replication services?

DELANEY: We offer three diskette sizes: 3-1/2", 5-1/4" and 8"; 1/2" and 1/4" tape and cartridges; 8 mm cartridges; CD-ROM and O-ROM.

DEVELOPER: What publication services are included?

DELANEY: Publication printing and binding cover the range of documentation to accompany software, from hardcopy or softcopy technical manuals to keyboard templates, diskette envelopes, custom labels and other support literature.

DEVELOPER: What other services are available?

DELANEY: We assemble the components of a software order into a complete, accurate package for software vendors and their customers. Additionally, we offer some specialized options. These include software and publication ordering, customer service follow-up and online technical manuals.

DEVELOPER: What new services do you envision for the future?

DELANEY: Depending on what customers want, we could provide electronic delivery, as well as software consulting services.

DEVELOPER: How can interested companies reach you?

DELANEY: That's easy. Just call our IBM sales team at this toll-free number: (800) 926-0364.

Vendor Support

SAA EXPRESS - Cooperative Processing for the 90's

by Kevin Kornfeld

Imagine this scenario: A software vendor has a host based application (MVS, VM or OS/400). The application is losing market share for a variety of reasons: it has a cumbersome user interface; it lacks graphic capabilities; it is unable to exploit user workstations; it lacks interoperability with other system platforms. In addition, workstations are an increasing market presence for the vendor's customers — yet few customers are prepared financially to give up their non-programmable terminals. To add to the confusion, many on the vendor programming staff have never programmed on a personal computer before.

SAA EXPRESS, a support program run by IBM's Software Vendor Operations (SVO), offers a solution to the above dilemma. SAA EXPRESS was established in June, 1990 to foster an on-going, working relationship between IBM and software vendors. SVO is split between three IBM locations: Milford, CT, the headquarters site responsible for marketing and program registration; Dallas, TX, with the technical support mission; and Atlanta, GA, charged with administration and contracts.

The goal of SAA EXPRESS is to accelerate software vendor development of Systems Application Architecture (SAA) cooperative processing applications, and enter these applications into the SVO "SAA Catalog", a yearly publication. For the vendor, it is a way of solving problem scenarios like the one described in a timely, efficient, cost effective, and quality manner.

Solving the problems of the vendor scenario (*see Figure 1*) is what SAA EXPRESS is all about. Using the OS/2 Presentation Manager and enabling tools such as EASEL™, Digitalk Smalltalk/V™, or Micro Focus COBOL/2™, along with a concentration on connectivity protocols such as EHLLAPI (Emulator High-Level Language Application Programming Interface) and APPC (Advanced Program to Program Communications), vendors are not only guided through the design and development of their workstation software, but also are shown how to interact with their existing mainframe applications via the workstation. With EHLLAPI, the host application doesn't need to change. Both the host and the workstation come together via cooperative processing, thereby meeting the needs of the vendor's own users.

How does SAA EXPRESS accomplish it's task? By providing a wide array of vendor services and offerings (*see Figure 2*), such as:

- A class on Cooperative Processing with OS/2 and EASEL
- A class on Cooperative Processing with CICS OS/2 and EASEL
- An Advanced Cooperative Processing class
- Question and Answer support via a 1-800 number and IBMLink
- Conference calls
- Strategy sessions

Kevin Kornfeld

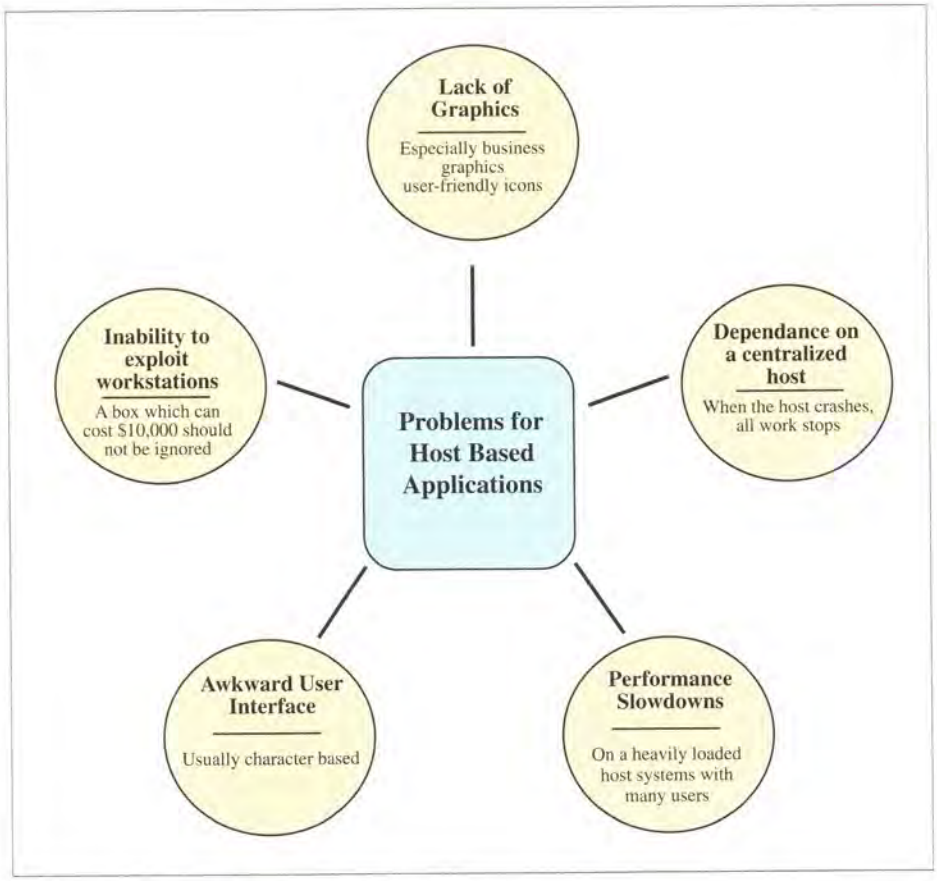


Figure 1. Some Typical Problems for Host-Based Applications

- Application walkthroughs
- Development Support Services
- A monthly newsletter
- Periodic teleconferences
- Loaner software of OS/2, the OS/2 toolkit, EASEL Workbench, CICS OS/2, Digitalk Smalltalk/V, or Micro Focus COBOL/2 Workbench
- Assistance with product marketing

EDUCATION

Cooperative Processing with OS/2 and EASEL is a one week class giving vendors an introduction to cooperative processing concepts in an SAA environment. The EASEL language is taught and used as a facilitating developmental tool. The class includes such subjects as:

- Cooperative Processing strategies
- EASEL Overview and Coding, including EASEL Workbench
- Introduction to SAA/CUA, Tips and Techniques
- OS/2 Presentation Manager Overview
- LU6.2 Communications Overview
- OS/2 Database Access
- Host Application Front-ending
- Inter-process Communications

The class is targeted for an audience wishing to learn how to establish cooperative processing applications between a programmable workstation (PWS) running OS/2 and an existing host application. Students should have previous programming experience as well as prior OS/2 exposure.

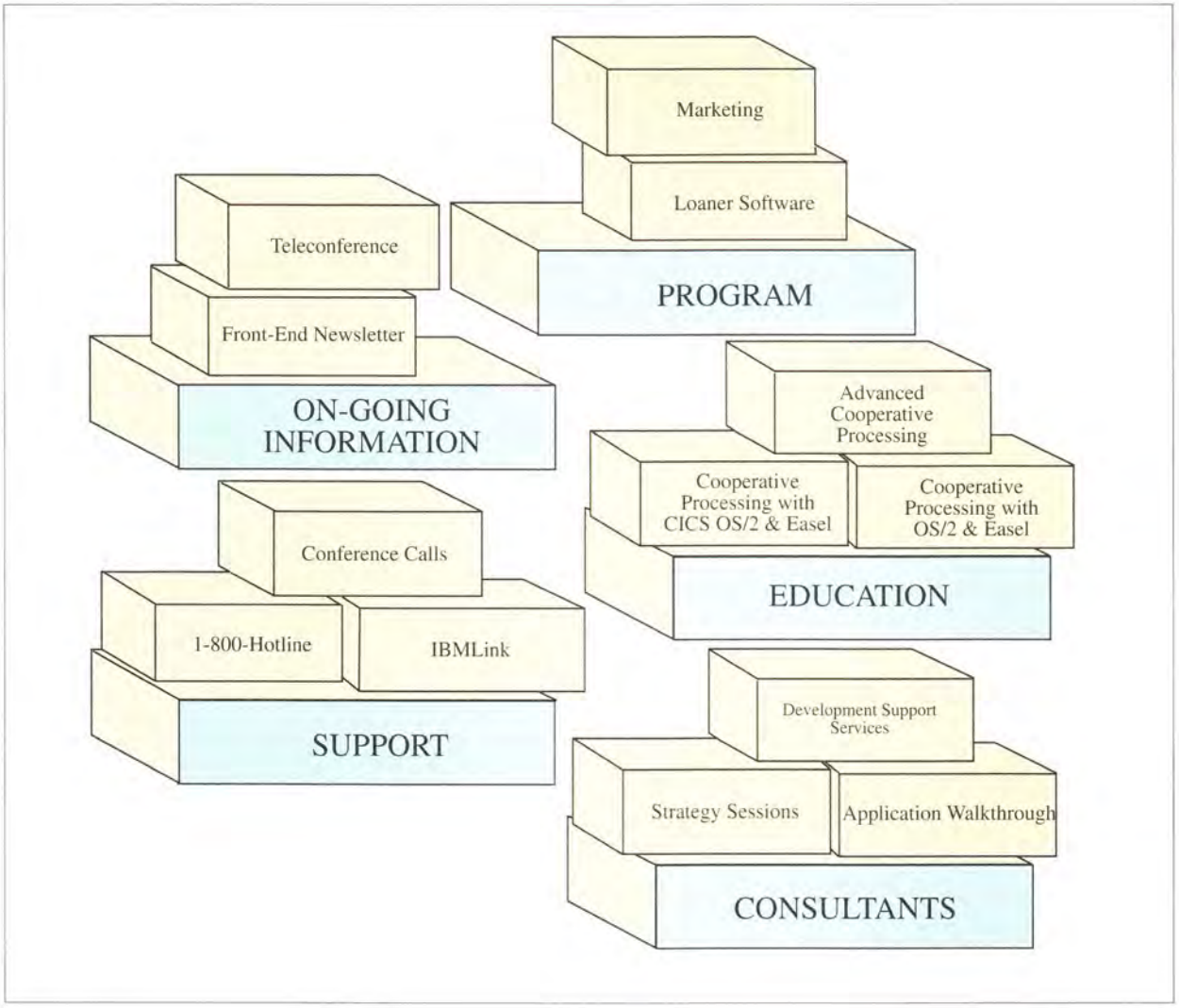


Figure 2. SAA EXPRESS Offerings

The *Cooperative Processing with CICS OS/2 and EASEL* class has a similar target audience and conceptual base, and also runs for one week.

In this class, however, the student focuses on CICS OS/2 as the communications and cooperative processing medium. The subject areas are:

- Cooperative Processing strategies
- Introduction to SAA/CUA, Tips and Techniques
- OS/2 Presentation Manager Overview
- CICS OS/2 Fundamentals
- CICS OS/2 Installation and Table Administration
- Interfacing CICS OS/2 to EASEL
- Distributing function and data with CICS OS/2
- Inter-process Communications
- EASEL Overview and Coding, including EASEL Workbench

For the *Advanced Cooperative Processing on OS/2* class, a slightly different approach is taken. Concentration is less focused on the user interface, although CUA '91 and object-oriented concepts ARE covered. Instead, there

is more emphasis on the technical issues involved in the distribution of function and data between host and workstation. Students are expected to know either "C", COBOL, or EASEL, and be familiar with CUA 1989 and OS/2. Subject areas for the advanced class include:

- Distributed Application Design
- CUA '91 Overview and Implementation Considerations
- Object-Oriented User Interfaces
- OS/2 2.0 Overview and the Workplace Shell
- LU6.2 Communications
- APPC & CPI-C Implementation Strategies
- Data Distribution

Upon enrollment in SAA EXPRESS, a vendor is allowed to send two students to any combination of two classes listed above. Vendors may send more than two students on a fee basis. Once education has been completed, vendors are encouraged to complete their SAA cooperative application within 12 months. To assist them in meeting this goal, support services are provided.

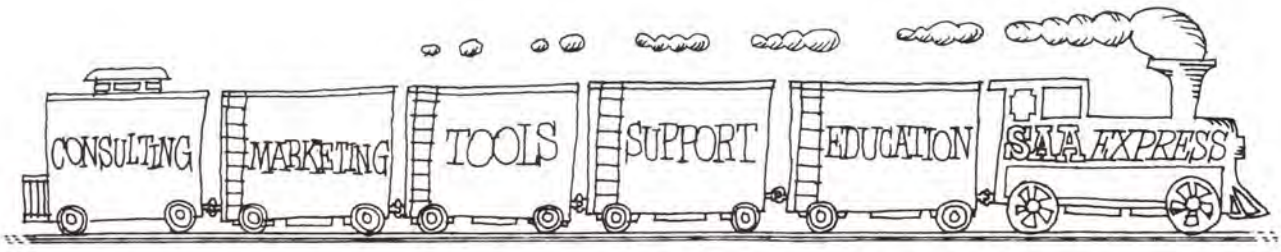
TECHNICAL SUPPORT

For typical programming or procedural problems that occur during development, a toll-free vendor support hotline and an electronic IBMLink id are provided. In this way, vendors can use the support vehicle of their choosing (telephone or computer) to receive a response to their query (almost always within twenty-four hours). For more detailed or complex issues, a conference call with several members of the Dallas support staff is usually available within a day.

STRATEGY SESSIONS AND APPLICATION WALKTHROUGHS

When technical or architectural questions arise at the start of (or, at key milestones within) project development cycles, a more fundamental evaluation can be provided to the vendor. This generally covers a wide range of issues, such as:

- Will code development be performed entirely in a CASE or 4GL tool?
- How can existing COBOL expertise in the organization be utilized?
- What is the most effective way to build up "C" and/or Presentation Manager expertise in the organization?



- Should APPC be used, and when? Can it be integrated with an EHLLAPI implementation? How?
- What is CPI-C, and when should it be used?
- Can data be distributed on a network? How? What options and/or methodologies exist for host-workstation data distribution?
- Exactly how does CUA relate on a per-screen basis to the existing host application?
- When should CICS-OS/2 be used? What advantages does it offer? How does it interact with EASEL, Micro Focus COBOL and OS/2 Presentaion Manager?
- What are some approaches and mechanisms for implementing multiple processes (or multi-threading) in a workstation application? Which functions might be implemented as separate processes?

Such a discussion is known as a "Strategy Session" and can center on any of the above subject areas, or around a variety of others. Strategy sessions are usually of maximum benefit at the beginning of project development, or at junctures of critical program milestones or decision points. Sessions typically last half a day, although full day meetings may sometimes be suggested. They can be performed via telephone conference call, video conference (facilities are available at many IBM sites throughout the United States), or on-site at IBM in Dallas, dependent upon vendor needs.

Another support option is that of the "Application Walkthrough", where an existing (or proposed) software architecture is analyzed by the technical support staff in Dallas. A walkthrough typically takes at least a full day, often two, and requires a significant time investment on the part of both IBM and the vendor. It is intended for providing significant review of a design proposal, and encompasses a detailed examination of design flows and interactions (along with feasibility \ and performance analyses). A formal recommendation letter is produced as a result of an application walkthrough.

DEVELOPMENT SUPPORT SERVICES

SAA EXPRESS also offers consulting services as two separate options: ExpressStart and Full Service Consulting. ExpresStart is offered following attendance at SAA EXPRESS education. It is designed for initial cooperative processing product development, lasts for five days, and involves working with the vendor development team on project start-up, preliminary architecture and design, screen development, and (possibly) a product prototype. Full Service Consulting is a customized contract designed to meet specific company requirements, and involves a full time consultant to assist the vendor in product design, development, implementation, and packaging.

NEWSLETTER

A monthly newsletter called "The Front-End" is provided to all SAA EXPRESS vendors, and contains articles and features designed to communicate relevant technical and program issues to vendors on an on-going basis. Past articles in "The Front-End" have covered a variety of subject areas, including:

- Dynamic Data Exchange
- OS/2 Performance Tools
- EASEL Workbench
- Creation of Dynamic Windows in EASEL
- AS/400 PC Support
- Dynamic Link Libraries
- Bitmap Image Formats
- Communication Server Routines
- Interprocess Communications
- Interfacing EASEL to CICS OS/2

TELECONFERENCES

Realizing that vendor education does not end with formal classes, a national teleconference is organized from time to time. This allows interested vendors (an average of twenty) to communicate with IBM and each other on a particular topic. Subjects have included "The Development of Large Applications with EASEL", "Benefits and Usage of Dynamic Data Exchange" and "Getting Started With EASEL Workbench".

LOANER SOFTWARE

An additional benefit for SAA EXPRESS members is the temporary loan of software, allowing them to better assess the long-term feasibility of their product decisions with some degree of capital expense relief. The software options include the latest level of OS/2 and the OS/2 Toolkit, CICS OS/2, and the System Performance Monitor/2, all for a loaner period of six months. In addition, there is an option of EASEL Workbench (three months), Micro Focus COBOL Workbench (including the CICS OS/2 option) (six months), or Digitalk Smalltalk/V (three months). After the loaner period, the software can be purchased at a discounted rate. Up to two copies of each product are available for loan.

PRODUCT MARKETING

SAA EXPRESS also offers vendors assistance with the marketing of their SAA applications, along with a listing of the application in the SVO "SAA Catalog" (published yearly). In September, 35 companies listed 56 cooperative applications available through the Catalog (via SAA EXPRESS).

VENDOR EXPERIENCES

More than eighty companies have enrolled to date in SAA EXPRESS, with each paying a $1500 fee. Many of these companies have already been successful. David Patrick, manager of DB/DC products for Aion Corporation of Palo Alto, California, told us, "SAA EXPRESS made it possible for us to deliver a graphical user interface front end to our product sooner than other alternatives."

Another program member is Erisco, a company of The Dun & Bradstreet Co., in Union, New Jersey. Bruce Kaurene, Senior Director of Health Care Information Services for Erisco, said, "The timing of the SAA EXPRESS program could not have been better for Erisco. The program has provided a great deal of support to our development effort."

A third company is Stockholder Systems Incorporated (SSI) from Norcross, Georgia. Comments from Dan Slay, a Data Processing Consultant for SSI: "The SAA EXPRESS program and EASEL helped SSI develop the Network Banker product about 30% faster than conventional development processes. Using PS/2 and OS/2 technology, the costs to install and operate the system are well below those of other bank-to-customer information delivery methods."

SAA EXPRESS is a program designed for today's software vendor to best meet the needs of tomorrow. For more information about SAA EXPRESS, contact Larry Sutton, Manager, at (800) 627-8363.

ACKNOWLEDGEMENTS

The author wishes to thank Andrew Aguilar for his assistance with the planning and editing of this article.

Kevin Kornfeld, *IBM Corporation, US Marketing and Services, 9 Village Circle, Roanoke, TX 76262. Mr. Kornfeld joined IBM in April of 1989 as an advisory marketing support representative. A member of the Software Vendor Systems Center, he works with select software vendors in developing SAA applications for the OS/2 platform, providing both design consulting and vendor education. Mr. Kornfeld has a BSEE degree from the University of Arizona in Tucson, and a Master of Computer Science from Southern Methodist University in Dallas, Texas.*

Software Tools

32-Bit Development Tools

by Brian Proffit

As OS/2® 2.0 becomes available, the demand for 32-bit tools continues to grow.

As this issue goes to press, the Limited Availability version of OS/2 2.0™ is beginning distribution, signalling the beginning of the 32-bit era—which many consider to be the key factor in the ascendence of OS/2. In order for that rise to take hold, though, the development community must have a quality set of development tools with which to exploit those new capabilities.

A sign of IBM®'s commitment to the developer is that there is now a department in Boca Raton, FL—OS/2 Tools and Technical Services—whose stated mission is to provide technical and migration support services to enable and encourage the development of a comprehensive set of application development tools, device drivers, and applications for OS/2.

By the time this issues reaches print, that department will presumably have made the following list nothing more than a small subset. Still, the following is a brief look at the 32-bit tools already available or announced for OS/2.

The primary tools for OS/2 2.0 development are the new IBM C Set/2™ compiler and Workframe/2™ developer's shell. (See IBM Application Development Tools for OS/2 2.0 , this issue, for more details.) An all-IBM product, the new compiler includes migration aids for those using IBM C/2™, Microsoft C 5.1®, or Microsoft C 6.0®. The optimization (which is safe at all levels) is world class. The diagnostics reach a new level of user friendliness. All of this, naturally, integrates seamlessly into Workframe/2™, the programmer's workbench.

Workframe/2 represents an environment similar to that found with compilers from other companies, in that it provides a shell within which the developer can work on individual modules and intelligently create an executable module by re-processing only the particular modules which changed. It provides for intelligent interaction between the compiler and the editor and the debugger.

Workframe/2 goes beyond this in several important ways, though. Perhaps the most important is that it is an open system. The interfaces by which Workframe/2 allows these interactions are published, and tool vendors are encouraged to write to those specifications. What this means is that the developer can select a Brand X editor, a Brand Y compiler, and a Brand Z debugger, and they will all interact seamlessly through the interfaces of Workframe/2. Borland International, for example, has already stated their commitment to the Workframe/2 APIs. This brings the developer the flexibility of selecting individual tools that match his tastes and needs without sacrificing the benefits of an integrated development environment.

One of the nicest new tools for OS/2 2.0 is the Enhanced Editor which the user may optionally include during OS/2 installation. It is a PM editor with a wide range of features. A new 32-bit version of REXX is also included in 2.0.

Brian Proffit

Other announced or available 32-bit development tools include:

Applications Manager, Intelligent Environments: An integrated GUI development environment using a proprietary 4GL to allow application development via point-and-click.

APS Series, InterSolv: A source code/application generator, including version control based on PVCS. Generates COBOL code which can be modified and compiled.

Arity/Prolog, Arity Corporation: A complete Prolog-based PM development environment, including support for Database Manager and LAN.

Borland C++, Borland International: An extremely high-speed compiler in an integrated development environment. Wide range of predefined objects and classes.

CASE:PM,™ Caseworks™: A family of interactive source code generators for C, C++, and COBOL.

Choreographer,® GUIdance Technologies, Inc.: Is an integrated GUI development environment using a proprietary OO language to simplify application development.

Common View, Glockenspiel, Ltd.: Is a C++ class library designed to simplify development of GUI applications.

Configurator/2, Configurator International: Is a networked C source code generator that includes a screen painter, diagrammatic program design, and tree-based program function description.

DataFlex, DataFlex Services, Ltd.: Is a portable 4GL RDBMS development tool.

DbxShield, The Stirling Group®: Greatly simplifies development of dialog boxes, and generates dialog box procedures.

DemoShield, The Stirling Group: Integrates screens from your program into a guided tour or tutorial, via demo scripting language.

ENFIN/2®, Enfin® Software Corporation: Object-oriented GUI application generator, including class libraries.

Glockenspiel C++, Glockenspiel, Ltd.: A C++ translator based on the AT&T specifications. Translates C++ code to IBM C Set/2.

Gpf, Gpf Systems, Inc.: An interactive C source code generator to simplify development of PM applications.

GUI_MASTER, Vleermuis Software Research: C++ class library for PM development, including over 80 GUI classes and an interactive interface builder from which new classes can be generated.

Guild,™ Expert-Ease Systems, Inc.: A highly graphical interactive object-oriented application generator.

Hamilton C Shell, Hamilton Laboratories: Over 130 UNIX®-like utilities designed to exploit OS/2.

InstallShield, The Stirling Group: Script-driven utility to allow the creation of PM-based installation procedures without programming.

LogShield, The Stirling Group: Macro record/playback utilities to assist in automation of application stress testing.

MemShield, The Stirling Group: Provides alternative heap management routines to simplify dynamic memory management.

Micro Focus COBOL/2 Compiler,™ Micro Focus: Complete environment supporting development of PM or host-based applications.

MultiScope, MultiScope, Inc: An interactive source-level debugger that exploits PM to allow multiple windows showing different views of the program and its data.

ObjectPM, Objective Solutions: Over 120 classes to simplify PM program development with C++ compilers.

ObjectVision, Borland International: Forms-based user development tool, allowing fast and simple development of applications without programming.

PolyAWK, InterSolv: UNIX pattern-matching language, including a pseudo compiler.

PolyMAKE, InterSolv: An intelligent make utility, designed to integrate with the PVCS version control system.

Primary Window Class, Software Engineering International: Supports development of CUA conforming object-oriented programs using plain C and Presentation Manager.®

PVCS Series for Configuration Management, InterSolv: Includes Version Manager, Configuration Builder, Professional Editor, and Production Gateway.

RM/COBOL-85, Liant Software Corporation: Portable COBOL compiler certified to 74 and 85 high standards.

Sage Professional Editor (SPE), InterSolv: Highly configurable editor with built-in emulation of several popular editors.

Smalltalk V/PM,® Digitalk, Inc.: Object-oriented development environment which fully exploits PM graphic capabilities.

Software Migration Kit, Micrografx® Corporation: A set of tools designed to simplify movement of Windows 2® applications to OS/2.

TbxShield, The Stirling Group: Simplifies creation of toolbox icon bars to improve user interfaces in complex applications.

TopSpeed®, C, C++, Modula-2, Pascal, JPI TopSpeed: A family of compilers which can share libraries.

VZ Programmer, VZ Corporation: An interactive object-oriented development environment. Many pre-defined objects.

Watcom C Professional Edition, Fortran 77, WATCOM: Optimizing ANSI compilers, including debugger.

Zortech C++, Symantec Corporation: Complete C++ development environment, including a wide range of pre-defined classes.

SUMMARY

This is quite a show of support for an operating system which is not yet generally available, and IBM is committed to making this list grow. If your company has an OS/2 development tool, or you have any questions or comments about OS/2 development tools, the Tools and Technical Services department can be contacted at 1000 NW 51st St., Internal Zip 2230, Boca Raton, FL, 33429. We also monitor the Application Development sections of the IBMOS2 forum on CompuServe.®

Happy developing!

Brian Proffit, *IBM Entry Systems Division, 1000 NW 51st Street, Boca Raton, FL, 33429. Mr. Proffit is a senior programmer in OS/2 software developer strategy. He is currently responsible for OS/2 developer tools. He joined the development laboratory in Boca Raton in 1983 after working as a systems engineer in Dallas.*

Software Tools

Introduction to OS/2 2.0 Application Development Tools

by Philip Winestone

That we live in interesting times is quite obvious. IBM®, as we all know, is making its own contribution to the tone of the times by introducing its powerful new, user-oriented operating system — OS/2® 2.0. Of course, this is only part of the whole story. Having produced this operating system — perhaps the most comprehensive operating system yet designed for the personal computer — we have to provide the means to design powerful, new 32-bit applications, and the means to migrate existing 16-bit applications to the 32-bit environment. Then, at last, we'll all be able to take advantage of the full power of the 80386- and 80486-based machines that we prize so dearly.

This is where we come in — the developers of the application development tools for OS/2 2.0, and this is what this article is about — the application development tools themselves. Our purpose is to inform you about, and perhaps, whet your appetite for the refreshingly new, comprehensive set of tools that we've developed. We would also like to underline our commitment to providing you, the application developer, with nothing less than the best of everything for OS/2 2.0.

Our commitment is to provide you, the application developer, with nothing less than the best of everything for OS/2 2.0

IBM C SET/2 VERSION 1.0

IBM C Set/2 Version 1.0 — IBM's 32-bit SAA™ C compiler — generates code for IBM OS/2 v2.0, and is designed to maximize the performance of applications using this new operating environment by fully exploiting the speed and power of 80386- and 80486-based computers. It comes complete with run-time libraries and a fully interactive, full-function, source-level Presentation Manager® (PM) debugger.

The 32-Bit C Compiler

By providing you with high-performance code optimization, the IBM 32-bit C compiler gives you the opportunity to produce some of the highest performing OS/2-based applications possible. IBM has focused intensely on such code optimization.

IBM also uses its register-linkage convention, OPTLINK, to improve performance, by using registers to pass parameters. Additional performance-enhancing features include:

- Improved memory management
- Inlining of selected library functions
- Memory file I/O support
- Fast floating point optimizations

During the compile operation, you can view error messages on three levels. LINT-like warning messages, grouped in the following subsets, can be turned on and off:

- Variables not explicitly initialized
- Non-portable code
- Appearance of goto statements
- Unused external variables

- Possible truncation during an assignment
- Preprocessor warning messages generated
- Parameters not used
- Enum message
- Preprocessor trace messages
- Name mapping

Standards

IBM's conformance to the following industry standards provides you with a standardized growth path:

- ANSI C X3.159-1989 and ISO/IEC 9899 C conformance
- SAA C CPI Level 2 conformance (excluding Record I/O)
- PM SAA/CUA™ conforming debugger interface

The IBM C Set/2 compiler implements the C element of the SAA Common Programming Interface (CPI) in the 32-bit OS/2 environment. For applications written to the SAA C CPI standard, you can readily export source code to and from the other SAA environments. The IBM C/370 and IBM SAA C/400 compilers, for example, implement the SAA C CPI standard on the System/370™, System/390™, and AS/400™ platforms respectively, as does the C compiler that comes with the AIX® Version 3 RISC System/6000™.

Here's how we ensure that your previous or current development tools are still viable.

- We provided 32/16-bit run-time coexistence and linkage conventions. You can call 16-bit interfaces, such as existing 16-bit libraries or APIs and make 16-bit callbacks to 32-bit code.
- We tailored the compiler for 80386 and 80486 processor technologies.
- We supported extensive, standard IBM run-time libraries to ensure consistency and continuity.
- We provided comprehensive migration support from IBM C/2 Version 1.1 and Microsoft® C 6.0-based applications.

Run-Time Libraries

We highly recommend that you use C Set/2's 32-bit compiler with IBM WorkFrame/2 which is shown in Figure 1. You can select your run-time libraries using WorkFrame's menu-driven compile options and you can mix and match libraries depending on the executable files you're trying to create. In addition, you can use the following libraries:

- Static and dynamic C run-time libraries
- Fully reentrant multitasking and single-tasking C run-time libraries
- Dynamic Link Libraries (DLLs)
- Subsystem development libraries

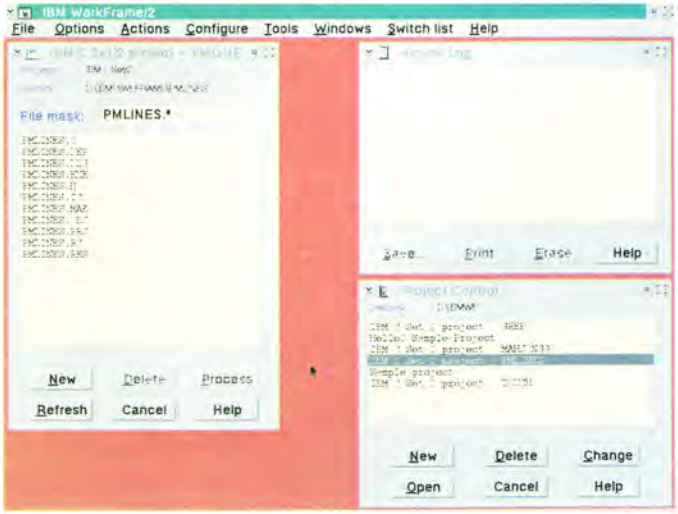


Figure 1. Using C Set/2 in the WorkFrame/2 Environment

Because the C Set/2 compiler gives you such extensive run-time library support, you can select the best execution environment for your OS/2-based applications. Not only does the IBM 32-bit C compiler let you use this considerable range of 32-bit run-time libraries, but you can also call 16-bit interfaces such as existing 16-bit libraries or APIs, and link all of these run-time libraries to your program either statically or dynamically. So, when you use C Set/2's 32-bit C compiler, you can still use the high-quality run-time libraries that you've developed over many projects.

In addition to the standard run-time libraries, C Set/2 offers libraries designed to help you migrate 16-bit C programs to the 32-bit C platform, as well as libraries for developing subsystems with no run-time environment.

Summary of Compiler and Library Features

- Multithread support for the C run-time libraries.
- Multitasking support for creating multiple processes.
- Static or dynamic linking of the run-time libraries.
- Fully reentrant libraries.
- Ability to create user dynamic link libraries.
- Subsystem development capabilities.
- Exploitation of 32-bit processor features.
- Ability to call 16-bit code from C Set/2 32-bit code.
- Memory files for temporary storage.
- Support of NaN, Infinity, and the 80-bit long double type as defined by Institute of Electrical and Electronics Engineers (IEEE).
- Support of the double-byte character set (DBCS).
- Compiler options specified on the command line or in an environment variable.
- Language level compiler options to simplify migration from the IBM C/2 and Microsoft C Version 6.0 products or to enforce SAA or ANSI standards.

The 32-Bit PM Debugger

Complementing the IBM 32-bit C compiler is the IBM PM debugger which features a graphical *point-and-shoot* Presentation Manager user interface. By making the debugger easy to use at the level of your mouse, keyboard and display, we have gone one major step further in taking some of the pain out of debugging.

Because debuggers are, by definition, user productivity tools, we have designed our PM debugger with advanced features such as Source Level Debug, Step-Mode Debug, and PM Application Debug Support.

Source Level Debug: You can follow the execution of a program in the compiler source view to quickly identify and correct errors in the code. You get a direct view into the execution of the program to help you identify errors earlier and correct them faster by the following methods:

- Stopping a program at user-selected breakpoints to monitor its progress and to find and correct program errors.
- Adding, saving, or deleting breakpoints or conditional breakpoints.
- Displaying and changing variables at breakpoints.
- Controlling execution of multiple threads individually.
- Evaluating expressions in the program.
- Monitoring variables, storage, expressions, stacks, and queues.
- Using multiple views of the program, including source and disassembly code.

Step Mode Debug: *Step and Go* gives you control to bypass previously debugged code and focus on problem areas.

Presentation Manager Application Debug Support: Synchronous and asynchronous modes give you two ways to debug PM applications. You can manage the application windows concurrently with the debugger windows.

Summary of Debugger Features

- Multiple program views, including source, disassembly, and mixed source and disassembly.
- Simple and complex breakpoint capabilities.
- Monitor windows for local, global, and automatic variables.
- Pointer and indirect referencing.
- Hierarchical structure display, including nested structures.
- Display of monitored variables in context.
- Ability to monitor storage and the call stack.
- Display of processor and math coprocessor registers.
- Support of DBCS.

Problem Determination Aids

In addition to the debugger, C Set/2 provides a number of problem determination aids which include:

- Debug memory management functions.
- Detection of possible programming errors using the /K*n* (LINT-like) diagnostic compiler options already mentioned.
- C source code listings. These listings include assembler listing code, expanded macros, and the layout of structures, as well as showing you what code the compiler has generated from your program.
- Precise diagnostic messages to aid problem analysis.

The C Set/2 Installation Program

You can install C Set/2 with the interactive installation program included on the C Set/2 diskettes. You'll find this very easy to do. Using the graphical interface, the installation program lets you choose the options you want to install and where you want to install them. Online help is provided throughout the program to help you with the installation. You can also use the installation program to reinstall or add new options later.

Online Documentation

Online documentation adds a new dimension to the compiler. It uses the Information Presentation Facility (IPF), which lets you view the information and *link* to additional information in a *hypertext* manner by selecting highlighted text. You can also search the online document for occurrences of specific words or phrases. These IPF features let you get to information quickly without affecting your workflow or your concentration.

Online Help

C Set/2 offers you an online reference and context and overview help for the debugger. The online reference contains information about declarations and definitions, preprocessor directives, include files, compiler options and messages, and library functions. You can access this reference through the View command. If you use the Enhanced Editor, you can get help for a particular item by placing the cursor on the word and pressing Ctrl-H.

With the contextual and overview help, you can get help for the various functions offered by C Set/2 debugger and help on how to use them. Access the debugger help from any debugger window by clicking on an item or from the Help pull-down.

C Set/2 Online Publications

C Set/2 provides online publications in two different formats: Information Presentation

Facility (IPF) Books and BookManager™ Books. IPF is the online help mechanism provided by the OS/2 operating system. The Online Reference (DDE4HELP.INF) is a summary of help for language constructs, library functions, and compiler options and messages. You can access the Online Reference from the command line. The system searches the Online Reference's table of contents and index, and if the item exists, opens the book at the panel about that item.

For example, typing

```
view DDE4HELP.INF operator precedence
```

opens the book at the panel about *operator precedence.*

Another simpler way is to use context-sensitive help via the Enhanced Editor. This editor uses a number of macros that enable it to provide context-sensitive help using the Online Reference. To obtain help for a keyword or construct, simply place the cursor on the problematic word and press Ctrl-H. The Online Reference opens to the panel for that construct.

BookManager READ/2 is an OS/2 product that allows you to read online documentation. BookManager features hypertext capabilities which allow you to link between related topics and search documents for keywords to quickly get at the information you need. The following publications are included with the C Set/2 in BookManager READ/2 format:

- *IBM C Set/2 User's Guide,* S10G-4444
- *IBM C Set/2 Migration Guide,* S10G-4445
- *SAA CPI Reference – Level 2,* SC09-1308

C Set/2 Hardcopy Publications

The publications that follow in Table 1 are provided with C Set/2. Each entry in the box contains the title of the publication, its order number, and a brief description.

IBM C Set/2 Publications

SAA CPI C Reference – Level 2, SC09-1308. Presents the SAA definition of the C programming language.

IBM C Set/2 Migration Guide, S10G-4445. Describes how to migrate IBM C/2 Version 1.1 and Microsoft C Version 6.0 programs to C Set/2 and contains a list of migration functions that the C Set/2 compiler supports.

IBM C Set/2 Debugger Tutorial, S10G-4447. Provides a tutorial for the C Set/2 debugger.

IBM C Set/2 and WorkFrame/2: An Integrated Development Environment, S10G-4449. Explains how to use the C Set/2 compiler and debugger in the WorkFrame/2 environment and provides a tutorial.

IBM C Set/2 Installation Card, S10G-4443. Describes the installation procedure.

IBM C Set/2 Reference Summary, S10G-4446. Summarizes the C Set/2 language syntax, keywords, library functions, and compiler options.

IBM C Set/2 License Information, S10G-4442. Provides a summary of the features and warranty information.

In addition, we recommend the following publication for those of you who are considering moving your C code from one platform to another.

Portability Guide for IBM C, SC09-1405. Describes how to move code from one platform to another, and how to write portable code. The second release of this document will include information on C Set/2.

Table 1 IBM C Set/2 Publication

IBM OS/2 2.0 DEVELOPER'S TOOLKIT

The OS/2 2.0 Developer's Toolkit contains a wide variety of language-independent build and productivity tools. You *must* use the Toolkit along with C Set/2 primarily because it contains the system linker (LINK386) that the compiler uses, as well as the system header files, the import libraries, and the NMAKE utility which increase the capabilities of the compiler.

The Toolkit includes headers and library files for defining OS/2 2.0 API calls, tools for building programs, API reference information, sample programs to demonstrate the coding of APIs, and a Kernel Debugger.

Build Tools

The Build Tools include the following utilities for working with executable files.

NMAKE: A utility that can save you time by automating the process of updating project files. NMAKE compares the modification dates for one set of files, the target files, to those of another set of files, the dependent files. If you have changed the dependent files more recently than the target files, NMAKE executes a specified series of commands.

You typically use NMAKE by specifying a project's executable files as target files and the project's source files as the dependent files. If you have changed any of the source files since the executable file was created, NMAKE can issue a command to assemble or compile the changed source files and link them into the executable file.

MKMSGF: Converts an error, help, prompt or general text information file to a binary format for display at runtime.

MSGBIND: Binds the message file as a data segment to the application's executable file. The MKMSGF and MSGBIND utilities help you isolate message text to support, for example, translation of text.

EXEHDR: Allows you to display or change fields in the header of an .EXE file.

MARKEXE: Marks .EXE files to indicate whether they can handle long file names and if they are window-compatible.

IMPLIB: Creates an import library. Import libraries are used to resolve an application's external references to subroutines in dynamic link libraries.

PACK: Compresses a file or group of files into a single file optimizing disk space and I/O performance on installation.

LINK386: Produces 32-bit .EXE files.

32-Bit PM Resource Tools

The 32-Bit PM Resource Tools include three resources editors, two compilers, and a precompiler.

DLGEDIT: A PM resource editor which lets you create and modify dialog boxes for use with other PM programs.

FONTEDIT: A PM resource editor which lets you create and modify fonts for use with other PM programs.

ICONEDIT: A PM resource editor which lets you create and modify icons, bit maps and pointers for use with other PM programs.

Information Presentation Facility Compiler (IPFC): Creates .HLP files for panels or viewable .INF files for documents from tagged text files.

32-Bit Resource Compiler (RC): Creates a binary file from Resource Editor (DLGBOX, FONTEDIT, ICONEDIT) output, making the resources available to the application.

System Object Model Precompiler (SOMC): Creates input to source programs that make existing classes available, and creates new classes of objects. (Bindings for C source programs are in the Toolkit.)

Libraries and Header File Directories

OS2286.LIB: Library against which system calls for 16-bit programs are resolved.

OS2386.LIB: Library against which system calls for 32-bit programs are resolved.

OS2STUB: Module included with an OS/2 program. At runtime, it alerts a DOS user that the program is an OS/2 program and therefore won't run on DOS.

OS2H: A directory containing ANSI conforming header files that define the OS/2 APIs for C.

OS2INC: A directory containing include files that define the OS/2 APIs for Assembler.

Online Information and Samples

Online information through the Information Presentation Facility (IPF) and sample programs help you exploit the OS/2 2.0 APIs in application programs. You can get additional guidance in the use of APIs, along with references to the sample programs, in the *Programming Guide*, the *Application Design Guide*, and the *SOM Reference;* all available in the OS/2 2.0 Technical Library (see below). Printed versions of the online API reference information are also in the Technical Library.

Debug Support

The Kernel Debugger is included in the Toolkit to assist with Ring 0 debugging. The Kernel Debugger is a set of files and utilities that you install separately from the tools and samples in the Toolkit.

OS/2 2.0 Developer's Toolkit Hardcopy Publications

The publications below provide information about the OS/2 2.0 operating system and the IBM OS/2 2.0 Developer's Toolkit. You can buy them individually or as the *IBM OS/2 2.0 Technical Library.*

OS/2 2.0 Developer's Toolkit Publications
Getting Started, S10G-6199. Presents an overview of the OS/2 2.0 Developer's Toolkit features and how to use them.
Application Design Guide, S10G-6260. Describes how to use the OS/2 2.0 Developer's Toolkit to develop applications.
Programming Guide, S10G-6261. Provides information on how to use the various features of the OS/2 2.0 Developer's Toolkit.
Information Presentation Facility Guide and Reference, S10G-6262. Describes how to create online help files for Presentation Manager.
System Object Model Guide and Reference, S10G-6309. Describes the object model used by the operating system.
Control Program Programming Reference, S10G-6263. Describes how to write control programs.
Presentation Manager Programming Reference Volume 1, S10G-6264. Presents the OS/2 functions.
Presentation Manager Programming Reference Volume 2, S10G-6265. Details the OS/2 function calls.
Presentation Manager Programming Reference Volume 3, S10G-6272. Describes window processing, messages, and other OS/2 features.
Physical Device Driver Reference, S10G-6266. Describes how to create and use physical device drivers.
Virtual Device Driver Reference, S10G-6310. Describes how to create and use virtual device drivers.

Table 2 OS/2 2.0 Developer Toolkit Publications (Continued)

Presentation Manager Driver Reference, S10G-6267. Provides information on developing drivers for devices operating in an OS/2 environment.

Procedures Language 2/REXX User's Guide, S10G-6269. An introduction to REXX features.

Procedures Language 2/REXX Reference, S10G-6268. Provides a summary of REXX features and APIs.

SAA Common User Access™ Guide to User Interface Design, SC34-4289. Provides information on CUA basic guidelines.

SAA Common User Access Advanced User Interface Design Guide. Provides advanced information on CUA design guidelines and tools.

Table 2 OS/2 2.0 Developer Toolkit Publications

IBM WORKFRAME/2 VERSION 1.0

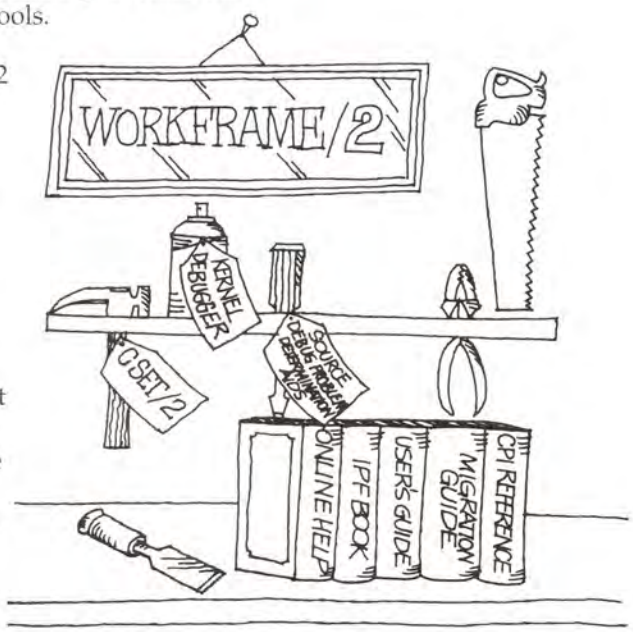
The PM-based WorkFrame/2 is complementary to C Set/2, giving you an open, highly configurable, project-oriented, language-independent development environment. You can easily customize the WorkFrame/2 interface to create a personalized environment by integrating your own choice of existing or new 16-bit and 32-bit OS/2 tools, as well as DOS and Windows tools.

WorkFrame/2 organizes your workplace by grouping your files into logical units referred to as *projects.* These can consist of both base and composite projects, the latter of which can be formed by combining a hierarchy of base and/or composite projects. Each of your projects can consist of source, object, and resource files, as well as targets such as .EXE or .DLL files.

WorkFrame/2 achieves this level of workplace organization by the following methods:

- Setting up projects to consist of source, object, and resource files and targets such as .EXE or .DLL files.
- Combining base projects to form composite project.
- Associating each project with your choice of unique compiler/debugger/maker/linker combination referred to as a *language profile.*
- Creating an Action Log to document all your actions, and their respective return codes, for every project that you initiate through WorkFrame/2.

You will find that if you use WorkFrame/2 as the integration medium for your development tools, you will almost certainly increase the productivity of these tools.



Because WorkFrame/2 lets you seamlessly plug in different or multiple edit/compile/debug tool components, you can choose whatever mix you like. In IBM's case, the edit component, the Enhanced Editor, must be installed if you want to use the Online Help facility. Note that the default editor in OS/2 2.0 is E and the default Compile/Debug component is IBM C Set/2.

WorkFrame/2 contains tools such as MAKEDEP and LIB. MAKEDEP is a Make File Creation Utility which creates the make file that NMAKE works with and is shown in Figure 2. This utility consists of a graphical interface and a language-independent DLL which interfaces to a DLL provided with the compiler.

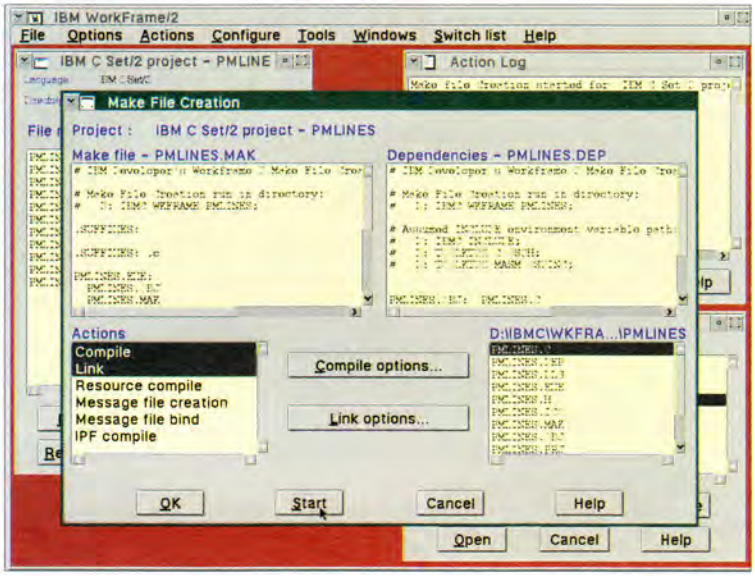


Figure 2. Creating a Make File using WorkFrame/2 MAKEDEP

LIB contains both graphical and command-line utilities which let you maintain object libraries. It can be used to do the following:

- Create a library.
- Add objects to a library.
- Unload objects to any directory.
- Change the characteristics of a library.
- Generate listing files of different levels of detail.
- Display the details of object modules.
- Delete objects from a library.

C Set/2 provides compiler option dialogs that you can use from the WorkFrame/2 environment, and includes a number of directories containing sample projects that demonstrate the capabilities of the WorkFrame/2 and C Set/2 combination. These sample projects include HELLO2, GREP, MAHJONGG, PMLINES and TOUCH. A tutorial guide is also included to help you use C Set/2 in the WorkFrame/2 context.

C Set/2 comes with a number of files that allow you to integrate it into WorkFrame/2. Among these are the C Set/2 Language Profile, which is used to associate WorkFrame/2 projects with C Set/2 and the Compiler Options DLL, which presents C Set/2 options through a graphical interface.

If you install WorkFrame/2 when you install C Set/2, the C Set/2 installation program creates project and control (.PRJ) files for the sample projects under the WorkFrame/2 home directory. The language profile and compiler options DLL are also copied to the WorkFrame/2 directory.

The many tools in the OS/2 2.0 Developer's Toolkit also integrate fully into WorkFrame/2.

HARDWARE, SOFTWARE AND OPERATING SYSTEM REQUIREMENTS

C Set/2 requires a system unit with a 32-bit processor (Intel 80386, 80386SX, or 80486) running the OS/2 2.0 operating system.

The IBM OS/2 2.0 Developer's Toolkit is a C Set/2 prerequisite, primarily because it contains the system linker, LINK386, that the compiler uses, as well as the system header files, the import libraries, and the NMAKE utility that increases the capabilities of the compiler.

To effectively use the compiler and debugger, you need a minimum of 8M bytes of RAM. A full installation of the C Set/2 files requires about 10M bytes of disk space, broken down in the following manner:

Compiler and libraries	**4M**
Debugger	**3M**
Online information	**2.2M**
WorkFrame/2 support	**0.8M**

If you do not have an 80486 processor, an 80387 math coprocessor is recommended because it will greatly increase the speed of floating point operations.

PRODUCT PACKAGING

You can purchase IBM C Set/2, IBM OS/2 2.0 Developer's Toolkit, and IBM WorkFrame/2 individually or combined in the *convenience kit* IBM C Developer's WorkSet/2, better known as *The WorkSet*, which is a complete C application development environment.

If you have your own compiler/debugger you may want to buy the OS/2 2.0 Developer's Toolkit and WorkFrame/2 in the other *convenience kit* format — the IBM Developer's Workbench for OS/2 2.0, commonly known as *The Workbench.*

Figure 3 shows the relationship between these two packaging options.

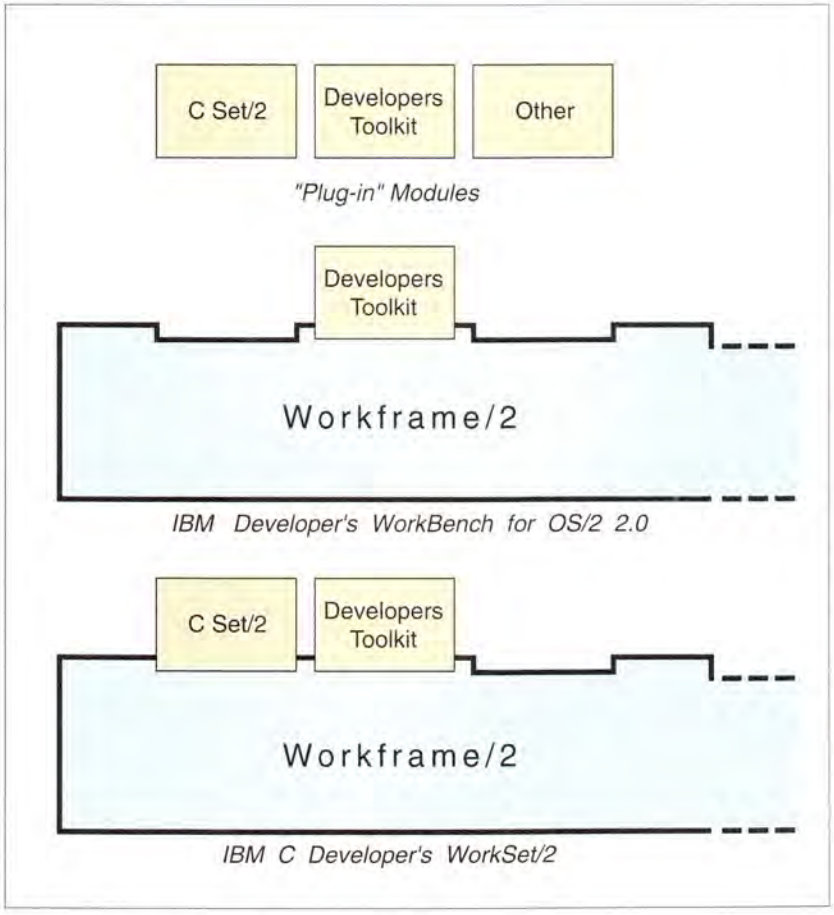


Figure 3. Product Packaging

SUMMARY OF PRODUCT PACKAGING

Name	Disk Size/Order Number	Category	Price
IBM C Set/2 Version 1.0	3.5/10G2996 5.25/10G3293	32-bit C compiler and debugger	Standalone Package – $695
IBM WorkFrame/2 Version 1.0	3.5/10G2994 5.25/10G3292	Language-independent, highly configurable, project-oriented environment	Standalone Package – $90
IBM OS/2 2.0 Developer's Toolkit	3.5/10G3355 5.25/10G4335	Language-independent, highly configurable, project-oriented environment	Standalone Package – $119
IBM C Developer's WorkSet/2	3.5/10G2995 5.25/10G3363	Three-product convenience kit	$895
IBM OS/2 2.0 Developer's Workbench	3.5/10G4333 5.25/10G4334	Two-product convenience kit	$199

Table 3 Summary of Product Packaging

Philip Winestone, *IBM Canada, 844 Don Mills Road, North York, Ontario M3C1V7.*
Mr. Winestone joined the IBM Toronto Laboratory in 1989 and is a Product Planner for OS/2 Application Development Tools. Formerly he owned a business in the imaging technology field. He received a BSC in Chemical Engineering from the University of Strathlyde in Glasgow, Scotland.

Software Tools

Open+Build™: A Voice, Fax, and Telecommunications Front-End for Databases

by Michael L. Fannin

Open+Build™ is an OS/2 application development and runtime environment, targeted specifically for the rapid development of voice and/or fax response applications. Open+Build is intended for use by application developers ranging from the corporate power user who is comfortable with PC database tools (such as Borland's dBASE or Paradox product, or the newer SQL products for LANS) to the professional systems developer who is routinely involved in computer system design and programming.

Open+Build can be thought of as an applications generator for a telecommunications front-end to a database. Instead of developing screen forms and keyboard interactive applications, Open+Build provides these either with TouchTone™, speech recognition, or fax as input with recorded voice responses text-to-speech, or fax as the response media.

Much like its screen-based cousins, Open+Build provides a range of tools for developing voice and fax response applications. The suite of tools include a visual development tool for quickly developing application scripts, a voice editor for preparing voice recordings, and a runtime engine for efficient execution and debugging.

WHAT ARE OPEN+BUILD APPLICATIONS?

Open+Build is used to build applications which provide telephone callers remote access to database information and database transactions. Callers may enter information and select options through the use of their TouchTone telephone. Responses may be provided to callers through voice recordings, text-to-speech, data transmissions, or fax responses.

Open+Build is typically used to build customized voice and fax response servers which are running group or department applications (GroupWare). Examples of these applications include voice response, fax back, remote order entry, price and availability. Other applications range from integrating voice and fax with existing applications to automating a call center.

Any business which has people using the telephone to provide information to callers is a candidate for an Open+Build application.

Michael L. Fannin

THE OPEN+BUILD ENVIRONMENT

Open+Build is an integrated development and runtime environment. Its typical application is to provide remote access to database transactions and information through a telephone. Open+Build consists of the following elements:

A Visual Case tool for rapid development

Visual Voice Editor

A runtime engine that:

- Supports PC, LAN, and mainframe databases.
- DLL connections to dBASE, Paradox, Btrieve, Netware SQL, IBM DBM, etc.

- A flexible, structured language with all the 'C' data types.
- Easy connection to other Dynamic Link Libraries (DLLs).

API support of a variety of voice and fax hardware adapters

Voice Adapters

- Dialogic
- Natural Microsystems
- Rhetroix
- Others

Fax Adapters

- Brooktrout Technologies
- GammaLink
- Others

Terminal Emulation Adapters

- DCA
- Others

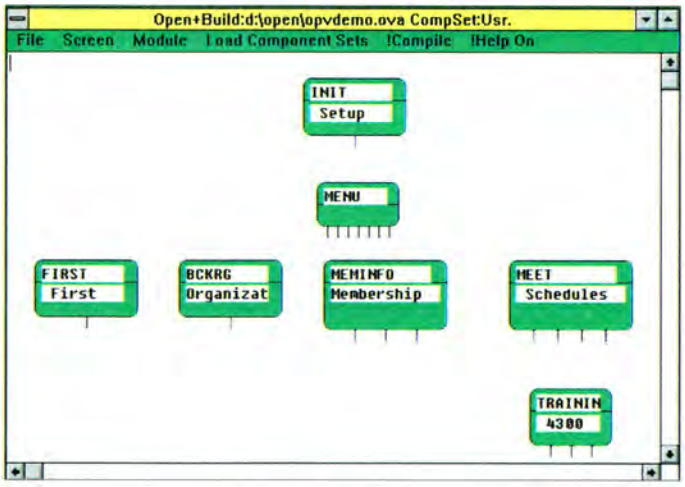
Details of the Visual Case Tool

This section introduces and defines several important terms and concepts which are used extensively by Open+Build: Network, Module, Components and Parameters.

Network

The network is the final completed user application. It is called a network because it consists of a group or network of modules. The modules of a particular application are arranged in a visual presentation by Open+Build to resemble a flowchart or block diagram. The network represents the visual and functional framework of an Open+Build application. See Figure 1 for a sample network layout.

A completed voice and/or fax response application is created by using Open+Build to design an application network. Open+Build may create new application networks, save completed or partially developed application networks in hard disk files, and retrieve previously developed application networks from a disk file. Open+Build can process only one application network at a time.



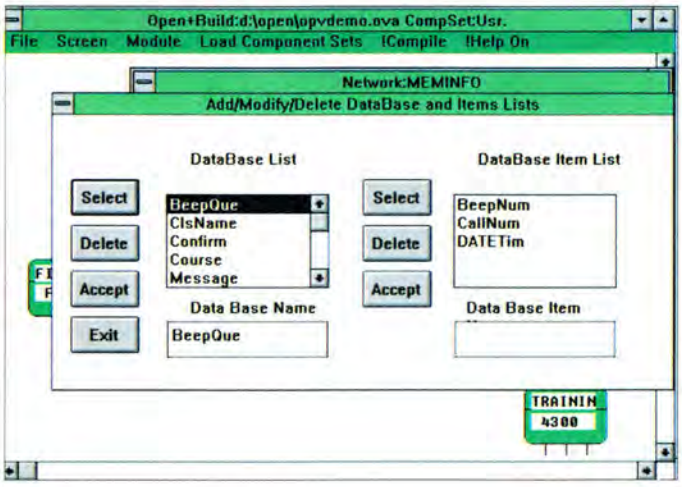


Figure 1. Screen shots from Visual Case Tool

Module

A network consists of a group of modules. Modules are subdivisions of the application network. Each module is assigned a particular purpose or task; all of the modules working together form the network.

The process of dividing an application into a network of modules is under complete control of the developer. The purpose of modules is to make implementation of applications easier and more understandable by dividing the target application into a group of smaller units. This modular approach to creating applications permits rapid development of new applications and makes it much easier to change the application in the future. The developer using Open+Build creates modules and arranges them in a visual display resembling a flowchart.

Modules consist of components. They are the primary work units, and a module's purpose and task are defined by choosing and configuring one or more component templates in a module.

Component

A component is the basic unit of functionality in Open+Build. One of Open+Build's more powerful features is its large selection of components. Each Open+Build component performs a specific task, such as: answer a call, speak the date to a caller, record a voice message from a caller, speak a previously recorded voice message to caller, or receive a fax call and store the fax on disk. Components are also provided to perform database transactions, look up information in a database, and create voice menus.

Types of Components

Open+Build consists of several sets of components and new components are being added every quarter. The following table shows the categories of components available in Open+Build and the number of components available in each category. (See Figure 2.)

Component Types	Number of Components
Standard Set	
1. Arithmetic/Decision Control	15
2. Database	11
3. Screen & Keyboard Control	4
4. Telephone Line Control	9
Script Set	33
Parameter Set	66
Accelerator Set	
1. Voice Message	9
2. Voice Prompts	7
3. String Components	10
4. Miscellaneous	3
5. InterTask Communication	4
Fax Set	15
Total Components	187

Figure 2. Component Family Organization

Parameters

Components are general purpose units of functionality. These units are configured from a template for a specific application and a specific purpose through data elements called parameters. Parameters are units of information which direct the components to perform actions specific to a user's application.

Parameters in these component templates fall into four categories:

Voice Prompts: These are prerecorded messages which are spoken to the caller. A typical voice prompt would be the oral greeting spoken to a caller when a call is answered, or the oral message spoken to a caller to present voice menu choices.

Database Definitions: These are the names of both database files and the data fields within these files. They are used by various components to access and update information in a database during an application. See Figure 1 for a sample dialog window used for creating and maintaining database definitions.

Variables: These are names of information cells that are used to store variable and temporary information used by the application. The information is called variable because it is not located in a disk file and it is typically very dynamic. A typical use of a variable would be to hold a series of TouchTone digits a caller dialed; another use would be to hold the current subtotal of an order a customer is entering by telephone.

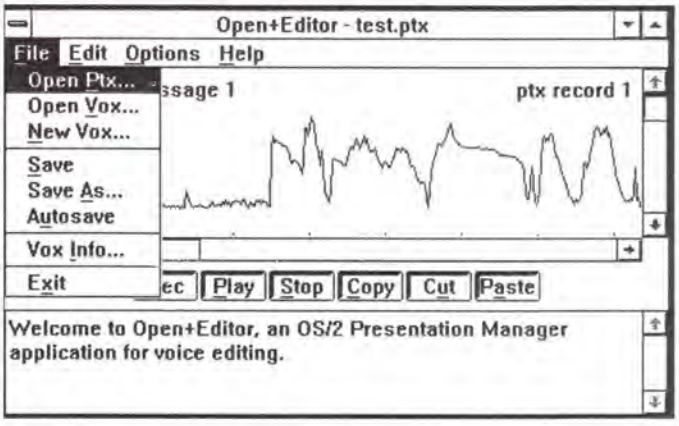


Figure 3. Visual Voice Editor

Constants: A constant is a special type of variable whose value never changes. It is typically used to initialize another variable to a predictable starting value, e.g., setting a subtotal equal to 0 at the start of a call.

VISUAL VOICE EDITOR

Open+Editor is the GUI controlled voice editor provided with the Open+Build environment. This editor is used to create and modify voice prompts used in the application. The editor is a PM based application with two modes of operation. It can be used on single voice files or it can manage a library of voice files. Other features include the ability to support a voice clipboard that enables the cutting, copying and pasting of voice segments from one file to another. (*See figure 3.*)

THE "DL LANGUAGE" ENGINE

The DL Language

The foundation language that underlies the Open+Build product is called the DL Language. It is an OS/2 product targeted for the development of integrated communication applications. It provides a high level, flexible interactive development environment specifically designed for the integration of various communication products, both hardware and software. An integrated communication application is any application using voice, fax, or data communication methods to access or interact with a data base or software system. Typical of these are voice response, predictive dialing, and document on demand fax back systems.

A runtime engine that:

- Supports PC and LAN-type databases.
- DLL connections to most communication expansion available for a PC.
- A flexible language, structured language with all the 'C' data types.
- Easy connection to other Dynamic Link Libraries.

The hardware vendors of these technologies typically supply AT or Micro Channel hardware along with a software driver and "C" language application program interface (API). These API's are packaged as OS/2 DLLs. Software vendors of various products also supply API's to their products which are packaged as OS/2 DLLs. Examples include OS/2 EE communication and database manager, OS/2 system services, MicroSoft's SQL server, Borland's Paradox and Asymetric's ToolBox product to name a few.

The typical professional developer writes a high level application in C and uses the DLL interfaces to access and control these hardware and software products to provide a unique application.

The purpose of the DL Language is to provide a high level, highly flexible interactive environment specifically designed for DLL connectivity and integration of various products, both hardware and software. It is uniquely suited to provide the specific logic flow of a unique application while dramatically reducing the development and test time typical of a low level language development method.

The DL Language has many of the features of a modern structured language such as a rich set of data types (ints, uints, longs, char, strings, arrays and structs), support of modules with private data scope and control constructs like If Then Else and Loop. A program is compiled into a compact tokenized format and executes applications using a core interpreter which runs as an OS/2 process. The language is specifically designed to provide the high level integration of multiple communication technologies through its ability to connect to any OS/2 implemented DLL. A DLL connection definition is a simple one line definition and the variety of data types of DL provide an ability to send and receive data to DLLs in virtually any data form.

Structure Overview of the DL Language

The DL system environment consists of four elements, a core interpretive engine, a set of vendor-independent script application commands, and a Presentation Manager® (PM™) window used to control, develop, test and interact with the development environment.

The core engine compiles and executes a specific application program. The engine can be interrogated through its API to retrieve static and running information regarding the application such as the current program position and contents of a data item. The script application commands are independent of the specific communications hardware or database software being used by the application.

Compiler

- Implements the basic language rules
- Compiles source code into a compact interpretive form

Runtime Interpreter

- Executes an application through interpretive process
- Provides realtime access and control of execution:
  - Current data objects contents
  - Single step
  - Trace Information

PM Control Window Description

The control window is the user's interface point into the system. It is a typical PM window with menu items to control various system actions. Features of the window include:

- Integrated text editor for creating source code.
- File control menu for loading source code and compiled program images.
- Compile and Test.

Example of DL Language

Figure 4 is an example of the structure and functional context of DL Language. The C-like structure is evident. The primary difference is that DL Language is interpreted, which is useful in developing communications applications, and it is designed to interface to most OS/2 DLLs. This provides an inherent open architecture and expendability to DL Language and, consequently, Open+Build, which uses DL Language as its foundation.

```
Preprocessor:
          INCLUDE <file>
          DEFINE <this> <asthis>
Global declarations (can appear before or after mainline
<statement>s):
           STRUCTURE <templatename>
                <type>|<templatename> <name>
                ....
          FUNCTION <funcname> ( [ <type> <name>, ....])
                <statement>
                ...
          IMPORT [CDECL]<dllname>:<entryname>([<type><name>,...])
          <type>|<templatename><name>(mainline vars are global)
<type> is one of:
          STRING
(var len string(auto decl))
          BOOL
      (TRUE or FALSE)
          BYTE
      (8 bit signed)
          UBYTE
   (8 bit unsigned)
          INT
         (16 bit signed (auto decl))
          UINT
      (16 bit unsigned)
          LONG
      (32 bit signed)
          ULONG
  (32 bit unsigned)
          HANDLE
(magic cookie)
Structure definition:
        <templateName> <structname>              (structure)
Array definition (at least 2 dimensions supported)
          <type> <arrayname> ( xsize [, ysize ] )
```

Figure 4. Overview of D Language (Continued)

```
<statement> is one of:
          IF <expr>
                    (start IF block (BOOL <expr>)
          ELSE
                                        (start ELSE block)
          ELSEIF <expr>
        (yet another IF (BOOL<expr>)
          LOOP
                                        (start of LOOP)
          BREAK
                                   (goto end of LOOP)
          CONTINUE
                         (go back to LOOP)
          PRINT <expr>, ...
 (simple print)
          INPUT <var>, ....
 (simple input)
          RETURN
                              (return from function)
          GOTO <label>
          (goto (yuk))
          <label>:
                         (labels for goto)
          <name> = <expr>
     (assignment)
          <function name> ([<expr>, ...])       (function or
import call)
<expr> has precedence with left-to-right eval:
          ()
(funcs,arrays,structs)
          - NOT SIZEOF HANDELOF STRUCTUREOF     (unary ops)
          * / MOD
                             (binary ops)
          + -
                  >> <<
                  <  >  <=  >=
        = <>
                  AND OR XOR
Constants:
                  1234 (decimal), 01234 (octal), 0x1234 (hex), 'A'
(byte)
                  "ABCD" (STRING)

        In strings: \n \r \t \b \f \\ \ooo
```

Figure 4. Overview of D Language

Script Application Commands

The Script Application Commands provide a set of standard, high level, voice, fax, telephone, and database functions which are implemented in a vendor independent manner. Some of these modules are written in the DL Language and some are DLLs written in C. Their purpose is to provide a tool kit of commonly needed functions which can be used to quickly develop an application. Open+Voice will be constantly adding to and improving this list of application commands. A developer may create an application by using modules only or may freely intermix modules and DL Language source code.

Support of a variety of Voice, Fax, and other Communications Hardware Adapters

Included in the basic product is the ability to support multiple hardware adapters. The device drivers for these adapters are supplied by the manufacturer, but the Open+Build and DL Language environment includes a set of API's which interface to these drivers and converts them to a common interface at the API level. At the present time the following voice and fax hardware adapters are supported: Dialogic, Natural MicroSystems, Rhetorix, Brooktrout Technologies, GammaLink, and certain DCA products.

The primary significance of the multiple hardware support is that users can select the best communications hardware for their application. They are not restricted to one vendor's product. Further, as new hardware is announced, the applications can be upgraded, thus protecting the users' investment in their application.

Support of a Variety of Application Databases

Just as Open+Build and DL Language use an open architecture philosophy to support a variety of hardware devices, the products also support a variety of application databases. Virtually any database that has its own or a third party's DLL is supported. This includes dBASE, Paradox, several vendors SQL databases, and numerous other databases. These databases can be read or written. Multiple types can be used concurrently in an application.

Mainframe databases are accessed through a terminal emulation process. This means virtually any mainframe database is supported.

The primary significance of the database support is that a communications front-end can be added to a "core" application without modifying that application.

TYPICAL APPLICATIONS

The following is a brief review of some applications that were developed with Open+Build. The time to develop these applications ranged from two days to two weeks.

Remote access to sales data

A major pharmaceutical manufacturer is using an Open+Build application to provide its field sales organization with access to the the customer order database. The objective of the application was to provide the field sales organization better visibility to customer ordering activity. An Open+Build voice front-end to the order database gives sales people instant access to the order database from any TouchTone telephone. They can track daily and month-to-date order activity. The system provides this information in the form of a voice response with an option to receive portions of the database or an immediate fax transmission.

Mailorder Customer Service

A sports clothing mailorder company wanted to provide automated access to backorder status for their customers. An Open+Build voice front-end to their back order database enables customers to check backorder status and ship dates 24 hours a day. The customers get a higher level of service, and the company's telephone agents are able to concentrate on processing new orders from new clients.

IBM Docu+fast™

IBM's Sales offices in Dallas, Houston, and Boston are currently using an Open+Build developed application which provides sales people and IBM business partners immediate fax back access to over 300 product brochures. From any TouchTone telephone an authorized caller can access Docu+Fast and select one or more documents to be immediately faxed to a customers. In addition to product brochures, items such as product configurations and training schedules are also available.

Mike Fannin, *Chairman, Open+Voice.*
Mr. Fannin is a pioneer in the voice processing industry with 14 years in the field. After a brief period in the OCR field, he has focused his career in the development of data and voice communication technology. He holds the industry's basic voice mail patent and successfully founded two voice processing companies. In June of 1978 he co-founded VMX Inc, the Richardson Texas voice mail manufacturing company, and in October 1983 he co-founded the Tigon Corporation, the Dallas base voice mail service company. Mr. Fannin graduated from the University of Texas at Arlington in 1970 with a BSEE . Open+Voice is his latest venture and was founded in July of 1990 to provide voice and fax response software products for workgroup applications.

Software Tools

TalkThru: Integrating OS/2 Asynchronous Connectivity

Austin J. Donaghy

by Austin J. Donaghy

Mention asynchronous communications to many people and they'll conjure up visions of slow speed modems, telephones, and unreliable connections. However, today's PS/2® systems and improved modems offer the user enhanced speed, reliability, and data integrity. Despite its earlier image, asynch communications now represents an effective methodology and comprises a considerable portion of communications in use today.

But the use of dial-up asynchronous communications remains small compared to the large number of directly connected or LAN-attached workstations. Further, supporting software often has done little to cooperate with other applications or communications tasks. Typically, the reasons have been DOS platform restrictions and the view that the asynchronous and 3270/5250 environments are incompatible.

TalkThru™ for OS/2® marries asynchronous and 3270/5250 connectivity under a common interface, permitting communications applications to be developed for one, the other, or both arenas, using REXX, EASEL™, C, Intelligent Environments Applications Manager™, and other platforms.

TALKTHRU BASICS

In early 1990, Software Corporation of America was approached by the Enterprise Alliance Group of IBM®, which is charged with finding OS/2-based solutions for major customers by building prototype applications. They were encountering corporate customers who not only used 3270 or 5250 terminals, but also a variety of other devices connected to various types of hosts, and required an asynchronous communications product that fitted into their basic toolset. Nine months later TalkThru for OS/2 was born, not just as an emulation product, but as a commercial software product geared to enhancing and integrating corporate connectivity under OS/2.

One of the keys to the success of OS/2 has been its communications strength. This is especially important now that a typical workplace may involve not only local multitasking, but also online and dial-up services, E-mail, and even direct connection to other systems, such as 2-wire hookup to a VAX®. Where in the past many applications were relegated to "run when machine available" status, TalkThru enables them to be accessed as needed.

How Does TalkThru Connect?

Under OS/2, we begin with the RS232 port supported by the COM02.SYS device driver and employing a user-supplied program to perform the emulation tasks (*See Figure 1*). Developers wishing to program to this interface should familiarize themselves with the I/O Subsystems and Device Drivers manual.

Under OS/2 ES, we also can get the asynchronous services which use the ASYNCDDx.SYS driver and a basic terminal emulator from the Communications Manager. The drawback here is that the

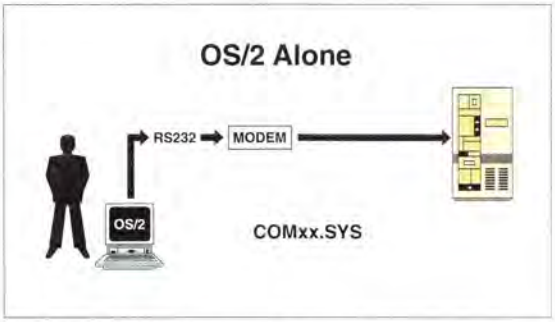


Figure 1. OS/2 Standard Asynchronous Communications

terminal selection is limited and only a partial implementation of the protocols has been used. A programming interface, ACDI, also has been supplied and documented in the ACDI Programming Reference manual. This facility also provides the flexibility of redirected I/O over a LAN using the ASYNCDDE.SYS device driver (*See Figure 2*).

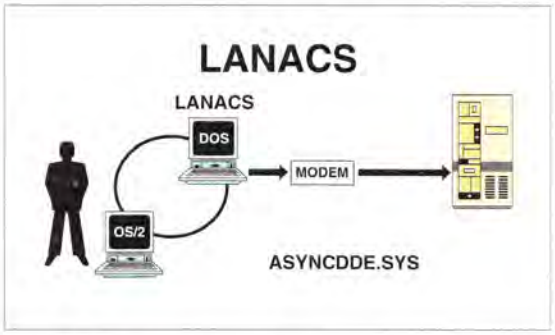


Figure 2. Local Area Network Asynchronous Communications Server (LANACS) implementation using the Communications Manager redirection device driver

Another connection choice is TCP/IP for OS/2 Telnet services utilizing user-supplied TTY and VT™100 emulators, or an outside vendor's emulator written to the system's Socket API. This API is documented in the TCP/IP for OS/2 Programmer's Reference (*See Figure 3*).

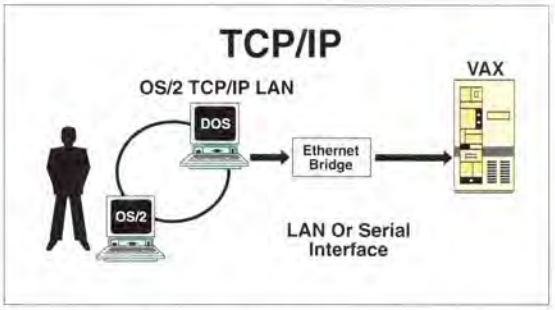


Figure 3. IBM TCP/IP for OS/2 Telnet Services Communications Paths

The advantage of these options is that a wide variety of connections can be supported. The disadvantage is that if all connection types are dotted throughout the organization, the user is faced with supporting at least three emulation products. Fortunately, there are common denominators here which can help guide the design of a common solution. Each of the connections provide a documented programming interface, each of them is used to achieve the same result (access to an asynchronous host), and the user community would like to see a consistent presentation regardless of the underlying technology.

The Roving Workstation

As a final note to the connection discussion, we should remember that the workstation may either be a desktop machine, a laptop, or a pen-based system. We now have machines capable of running advanced applications and also capable of being used both in and out of the office. And there's the rub — in and out of the office — two potential communications needs for one application. This typically means a change in emulation software so that the host may be accessed through a protocol converter, but automated applications must handle both environments. The requirement here is the same as before, a consistent interface will let the application run successfully in both environments.

INTERFACE CONSIDERATIONS

The 3270 and 5250 applications have a common interface in EHLLAPI, but asynchronous applications have to be able to write to multiple interfaces to accomplish the same task. In TalkThru, one of the goals was the marriage of the two technologies under the one EHLLAPI interface. We had two choices: either reduce 3270 access to a lower common denominator, or attempt to raise asynchronous protocols to a higher plain. Both of these involve some compromise, but our choice of the latter option proved to have minimal impact in many situations. Incorporating EHLLAPI into TalkThru

enables users to develop communications-cooperative applications for multiple hosts using the tools of choice available in the marketplace (*See Figure 4*).

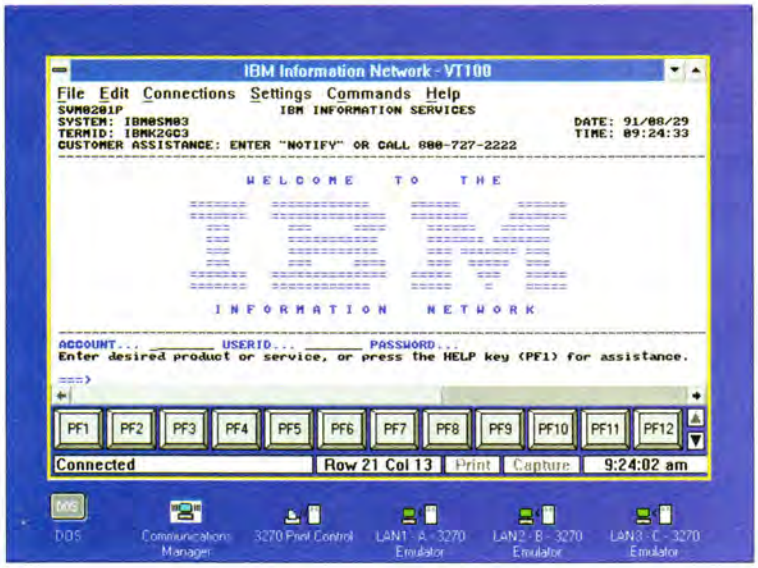


Figure 4. The TalkThru for OS/2 Terminal Emulator Window

Using The Features of OS/2

Presentation Manager® provides us with a rich display vehicle incorporating such features as fonts, colors, sizes, and placement. Fonts are important items to consider in terminal emulation because many devices incorporate their own character sets, especially foreign and line drawing characters. OS/2 permits us to implement our own fonts and we also can supplement them with user or system fonts interpreting special characters on a best-match basis. Color is always a matter of personal preference. Certainly, a PM application must honor the system-wide color options employed by the user. However, because of the variety available in some advanced protocols, it would be nice to be able to tailor all color aspects of the screen.

The size and placement of the terminal screen is a simple function of its being a PM window. Our goal was to ensure that a 24x80 screen could be contained within the workplace and not be the entire desktop. We combined this with multiple font sizes to satisfy individual user preferences.

Next we considered the value of multitasking and the capabilities provided for both foreground and background activity. To capitalize on this, each terminal application is a separate process, and can run visible, minimized, or totally hidden.

Once the various aspects of the end user interface were settled, designers were then free to exploit all facets of OS/2 process control, storage management, and interprocess communication. The main point is that we elected to write this application from scratch rather than port existing DOS or Windows® code. We then were free to use all of the features available in the operating system without worrying whether we would impact future development in another environment.

PRODUCT ARCHITECTURE

To create an Asynchronous Manager, we need a central point from which user selected tasks may be dispatched. It must present the user with a readily recognizable list of available sessions (which is preferable to choosing an emulator and then selecting where and how to call). One could simply use the Workplace Shell to chain off a selection list that invokes each potential session. This has the perceived advantage of maintaining basic "look and feel", but loses any relationship among multiple sessions by not having any central point processor. The latter, as we will see, becomes more desirable as we begin to integrate all components.

In TalkThru, our solution to the central point is the Phone Book (*see Figure 5*) with a menu bar that enables the user to add, change and delete entries, access sibling windows, and create new sibling windows. Each entry depicts a connection, whether this involves

Figure 5. The TalkThru for OS/2 Phone Book Window

a dial-up access or some other more sophisticated communication method. The user now has a descriptive, pictorial representation of the available selections. The icons are totally user definable and, as with the 2.0 Workplace Shell, also may be user created.

The Emulator

Building a terminal emulator can be defined as a fairly straightforward task, provided that the access method and protocol have been pre-determined and are not expected to change significantly. To provide more flexibility, we decided to design our emulator in three parts. First, we provide a standard routine to handle painting the window and responding to keyboard and mouse requests. Next, we define an API to permit screen and input activity to occur regardless of protocol or connection type. Then we separate those functions that are protocol related from those that maintain and service the actual connection. We then fill the two holes, one with protocol functions and the other with connection functions.

The reason we discuss this aspect of our design is to highlight the use of a very powerful feature of OS/2 — the Dynamic Link Library. By using DLLs we are able to optimize the product by separating activities into very specific functions that can be demand loaded to handle the current task, but will also service another process using the same facilities (*See Figure 6*).

Beyond the "plug and play" capability for protocols and connections which the DLL facility provides our emulator, we also use it to support file transfers. Asynchronous communications by nature access diverse hosts which, in turn, employ diverse file transfer methods.

Advanced PM Usage

A significant portion of asynchronous communications service is protocol conversion of 3270 emulation for dial-up access from a portable. While some features were intended to improve ease of use for the less sophisticated user, they increase the power available to the expert as well.

The HP® terminal type includes, as part of the display, images indicating the action currently assigned to a function key. In this emulation type, these values may be dynamically set from the host. Obviously, to complete the emulation of this device, we had to include this capability in our software. However, courtesy of the power of PM, we were able to implement this facility in the form of "buttons". Thus in a 3270 protocol conversion situation, we are able to define the

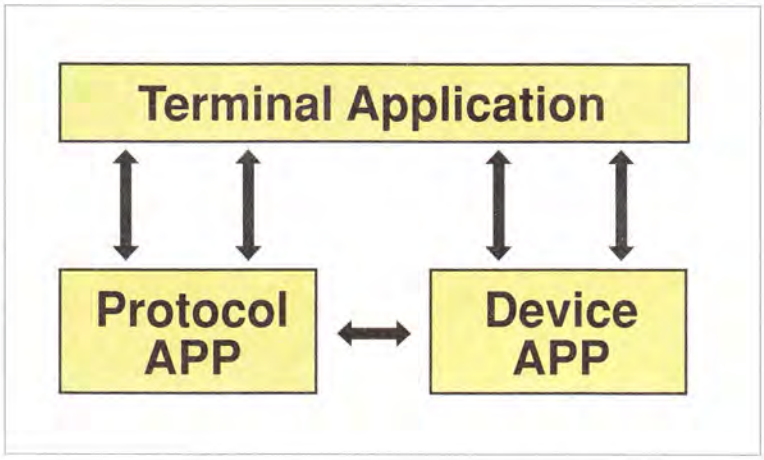


Figure 6. Separating Device and Protocol Functions into Independent DLLs Provides Better Mix and Match Capabilities

buttons to PF or system keys, giving us the ability to drive an application such as PROFS® for the most part via the mouse. In the same way, buttons may also be used to invoke other programs or communications tasks.

Another strength of TalkThru is the ability to assign various screen input activities to the mouse. For example, double clicking on a word could be used to transmit the indicated text with or without an appended enter key. The screen text may indicate that a particular key will perform a function, and once again we can turn to PROFS for an example. Most PROFS screens indicate something like "PF2 - Open the Mail", suggesting to the user that pressing that key will perform the desired function. TalkThru provides the ability to point at "PF2" and click the mouse to activate that sequence. Other settings may be used to position the cursor, etc.

COOPERATING WITH COMMUNICATIONS MANAGER

To date, asynchronous products have tended to be insular, talking only within themselves and generally ignorant of other communications sessions active in the machine. This was not unreasonable given

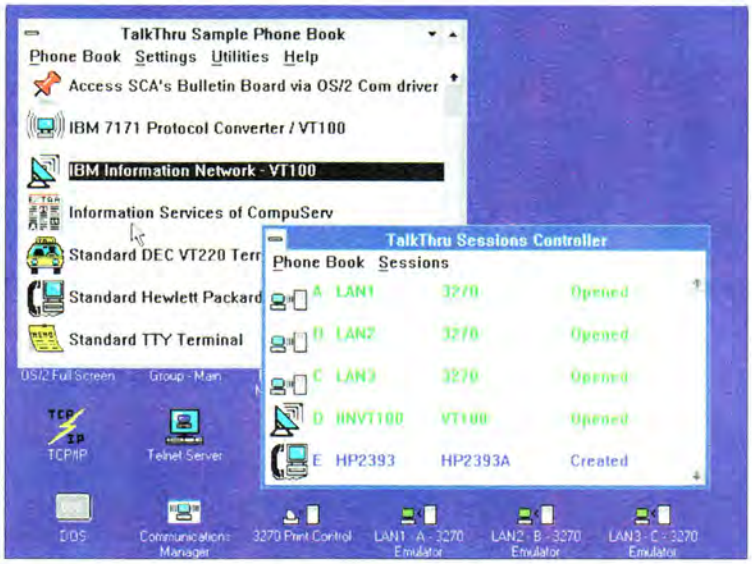


Figure 7. The TalkThru for OS/2 Controller Window Showing Integrated 3270 and Asynchronous Sessions

the operating environment, DOS, and the hardware limitations of the day. The problem often was compounded by oversize programs that necessitated rolling out or other such gyrations. However, such restrictions are behind us and we can now build applications that legitimately coexist with each other (*See Figure 7*).

When you create a product that cooperates with other communications players in the same OS/2 workstation, it is reasonable to assume that task automation is likely to be the driving force. If all you want your communications products for is terminal emulation, then simply fire them up and begin emulating. However, if you recall, we noted that IBM's Enterprise Alliance uses a powerful development tool, EASEL, to build communications-related applications. EASEL has excellent syntax for developing front end applications in a 3270/5250 environment. It does have some limited asynchronous capability, but implementation curtails some of the power of the 3270/5250 facilities of the product.

Enterprise Alliance, aware that some sophisticated asynchronous protocols have screen field attributes, required an emulator that could pass that information to the EASEL applications they were developing. To maintain their basic mandate and to include the other vendor systems, they sought an asynchronous product that could provide EHLLAPI service for diverse connections - particularly RS232, LANACS and TCP/IP. If you consider the potential variety of connections, along with the fact that most have unique APIs, there is obvious value in providing an alternative that presents all of these capabilities in a common manner.

EHLLAPI as a Common API

We have referred to the EHLLAPI interface a number of times in this article. For the benefit of those readers who are not familiar with it, following are its basic features. EHLLAPI is a programming interface that provides read/write access from within an application to 3270 or 5250 sessions active on the system. This facility lets you retrieve all or portions of the screen, referred to as the presentation space, either as a flat string or on a field-by-field basis. Applications may send text characters or sequences that represent the special keys of a 3270 or 5250 terminal. Various status information may be obtained via an API and examined to determine terminal status, particularly as indicated in the operator information area of a real device. Access to this API has now been provided in REXX under OS/2 2.0.

Implementing an EHLLAPI Traffic Cop

EHLLAPI is accessed from an application by making calls to an entry point in a Communications Manager DLL called ACS3EHAP. We created a filter to front end this module separating out 3270/5250 calls from those directed at asynchronous sessions. We also had to sidestep OS/2's active DLL naming rules.

The end user wants to communicate through an application, not a terminal.

The biggest drawback of this implementation is that you leave yourself open to being the first one called when a problem arises. You must make a commitment to accept this situation, and provide some verification capability for resolving problems. In TalkThru we provide the EHLLAPI Interface Tester, which permits a user, or our support staff guiding a user, to test any EHLLAPI situation (*See Figure 8*). Such tests may be conducted

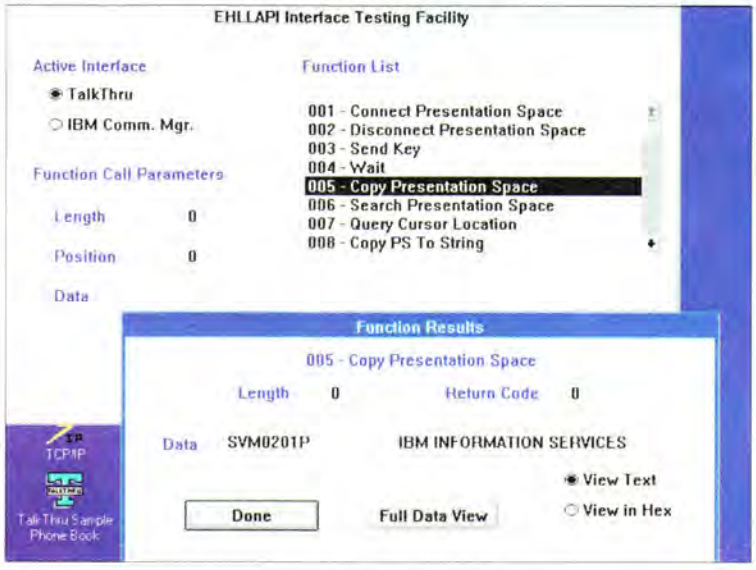


Figure 8. The TalkThru for OS/2 EHLLAPI Interface Testing Facility

using our DLL or eliminating it from the process by selecting the underlying Communication Manager DLL directly.

THE PROTOCOL PROBLEM

Earlier we mentioned possible compromises in implementing a programming interface that supports the 3270, 5250, and asynchronous environments. Most specifically this boils down to "what does the terminal emulator know, and when does it know it?" This is a matter of protocol.

Regardless of protocol, the terminal screen represents a presentation space. For many emulation modes, this presentation space has little value beyond a 24x80 buffer which may be examined for text content only. In the simpler protocol implementations, there are no field delineations or attributes and, therefore, EHLLAPI functions are limited due to lack of available information. Second, the less sophisticated protocols do not provide any mechanism by which a keyboard lock or input-inhibited state may be determined. The most viable implementation of an EHLLAPI application in these situations is one that has been created by someone familiar with the host systems, someone who can identify when the full presentation space is available and, then, that input may commence. This may sound very restrictive, but, in fact, very complex EHLLAPI front ends may be built even for systems as straightforward as TTY Information Retrieval Services or Bulletin Boards. An example is the EASEL weather station mentioned at the end of this article and shown in Figure 10.

Having considered the shortcomings of EHLLAPI in the asynchronous environment, let us examine the various protocols available, and how some of those provide a feature-rich environment for the EHLLAPI programmer. Before proceeding, however, we need to categorize the protocols into two types: character mode and block mode.

Character Mode

Character mode protocols are those where the host only sends enough information to paint the screen, generally echoes every keystroke, totally controls cursor positioning, and in no way indicates where input areas are until necessary. Examples of these protocols are TTY, VT52, VT100, VT220, HP2xxx in character mode, and ANSI. Another good example of this category is the variety of protocol converters available to IBM hosts, e.g. 3710, 7171, and various other vendor products.

Block Mode

Block mode protocols are those which operate in a manner not unlike a 3270 terminal. The host, in essence, transmits a data stream that not only contains the text to be displayed but also input field definitions that include attribute and size. These protocols also perform keyboard locking and unlocking as well as local cursor control. HP and TYMNET™ are examples of block emulation protocols which are superbly suited to the EHLLAPI environment. There are, however, some differences from a 3270 that require the EHLLAPI programmer to be alert in handling some situations. For example, the syndrome of Keyboard Lock Flickering (where the input-inhibited status in the Operator Information Area of a 3270 turns on and off between screens) is accentuated in the asynchronous environment, especially at slower speeds.

Putting This All Together

The result is that we now have a single communications environment that can integrate 3270, 5250, DEC® and/or HP emulation, as well as support mobile users whose requirements may be 3270 or asynchronous depending on location. The beauty of this model is the ability to create applications for all environments with economy of effort, utilizing a common API (*See Figure 9*).

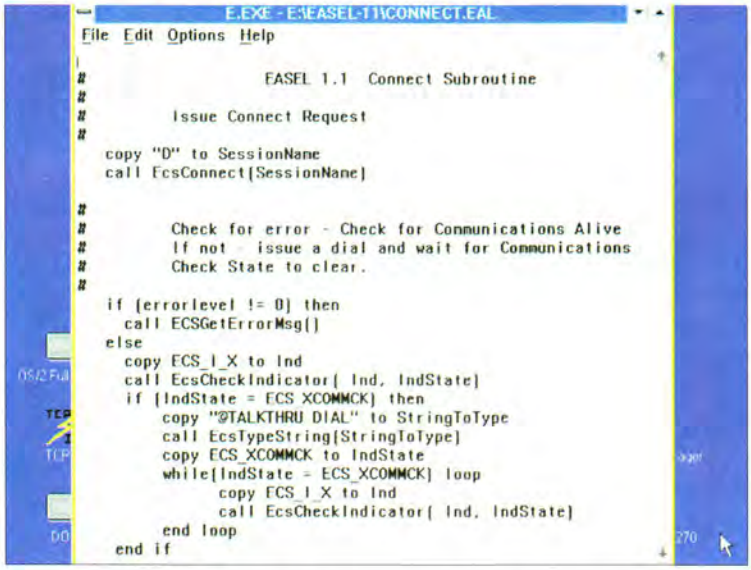


Figure 9. Sample EASEL EHLLAPI Code Using TalkThru to Dial a Phone

The application development tools in use today are 'C' programs, REXX, EASEL, and Applications Manager, to name a few. Included in the TalkThru product is another application development facility called AUTOPILOT. This is a communications-oriented language with which users are capable of developing sophisticated dialog based applications. AUTOPILOT overcomes the vagaries of diverse communications environments.

SUMMARY

To illustrate the power of this integrated communications platform, we refer you to the "weather station" shown in Figure 10. This is a snapshot of an EASEL application created by Andy Ellicott of Easel Corporation. The picture shows the result of a weather inquiry which the user made simply by clicking the desired city button on a US mainland map.

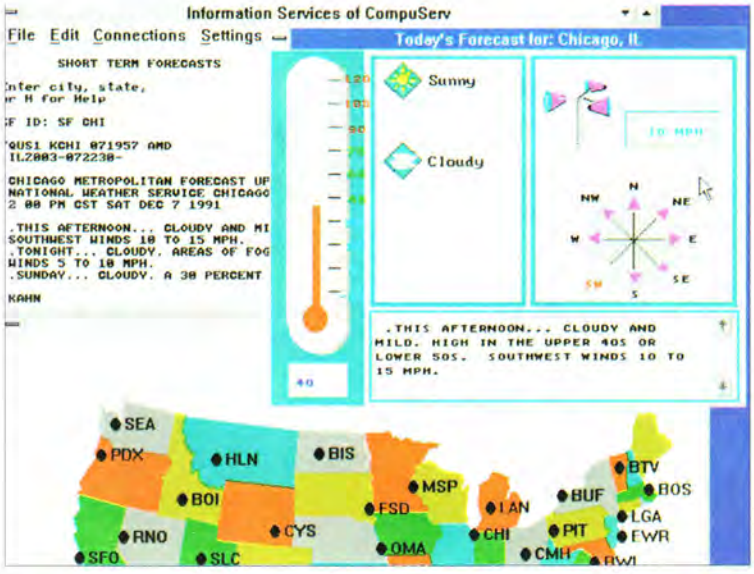


Figure 10. An EASEL Application for Obtaining Weather Information from CompuServe via EHLLAPI

The significance of this example is that the EASEL program was written to the EHLLAPI interface but is actually accessing CompuServe® as a VT100 over a dial-up connection.

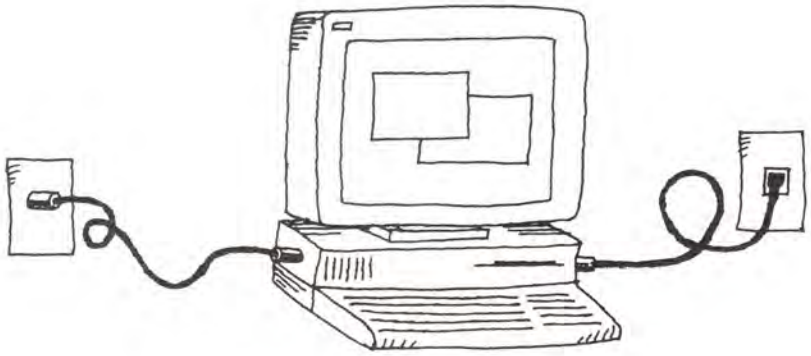
More and more, we see the end user's desire to communicate through an application rather than as a terminal. TalkThru expands the choices available to developers who are addressing that desire by adding to, and cooperating with Communications Manager and other OS/2 applications. We hope that by providing a unified communications front end, we have made the task of integrating diverse host environments, and the data available on them, less complicated and less daunting.

Austin Donaghy, *Software Corporation of America, 100 Prospect Street, Stamford, CT 06901, (203) 359-2773. Mr. Donaghy currently is Director of Product Development. Prior to joining SCA, he held systems-related positions within the life insurance industry. He joined SCA in 1984 and was one of the initial members of its New York-based PC development team. During his 18 years of computer systems development he has specialized in communications and operating systems.*

REFERENCES

IBM OS/2 Extended Edition V1.3 ACDI Programming Reference (64F3039).

IBM OS/2 Programming and Tools Information (64F0282).



Software Tools

SmallTalk/V PM: Getting Started in Object-Oriented Programming

by Gerry Strope

Getting started in object-oriented programming (OOP) can be difficult because of new concepts and terminology. Programmers used to the discipline of procedural languages such as Pascal, C, PL/I and REXX may have trouble relating to these new concepts. This article attempts to overcome these barriers by defining OOP in terms of procedural language examples. It does this by showing how to get started in OOP using Digitalk Smalltalk/V®PM, an object-oriented programming system produced by Digitalk Inc.

The following topics will be covered:

- Why a programmer should consider using OOP
- An introduction to OOP terminology
- Developing an application using OOP
- Digitalk Smalltalk/V PM on OS/2®
- The advantages of Digitalk Smalltalk over 'C' and Presentation Manager®
- Added function in the latest Digitalk Smalltalk/V PM releases

Before we get into OOP itself, we'll discuss some advantages gained by programming using OOP principles.

WHY USE OOP?

- Allows re-use of code

You could look at OOP as a set of building blocks. In OOP these building blocks are the objects. The OOP programmer is encouraged to create new building blocks that are reusable by adding and making changes to existing objects. Most OOP programmers are always trying to think of ways to generalize a piece of code to make it useful to themselves and others. What you typically will find in an OOP language like Digitalk Smalltalk (one that has been around for a while) is that all of the building blocks you will need to implement your design are already there. Over time, programmers have continued to add reusable objects to the environment.

- Keeps you out of the details

OOP programming gets you away from low level problems like defining control blocks, allocating and freeing memory, and learning complicated graphic functions.

- Less bulk

OOP programs tend to be less than half the size of procedural programs. Less bulk means less code to support and debug. The downside that needs to be considered is that the generated code tends to be larger.

Gerry Strope

Smalltalk/V is a very useful tool for learning OOP

INTRODUCTION TO OOP TERMS

If you go to a bookstore and buy an OOP book you will find standard definitions of OOP terms. If you are new to OOP these definitions can be hard to understand because they assume a certain level of knowledge about OOP. We'll explain some common OOP terms in relation to procedural languages. (OOP experts: forgive me for what may sound like an oversimplification of the OOP world. This article is targeted at the OOP novice and we are trying to de-mystify the subject!)

Object

An object represents a "thing" with a prescribed set of actions that can be used to get work done. Example: An array is something that all programmers have heard of. An array is an object in OOP language.

Class

A class name identifies the behavior of an object. Given a class name, the OOP environment knows how to create an object of that class. Example: "myArray := Array new: 20" would tell the array class to create an array of size 20. Now myArray is an array object ready to have things put into it.

Instance

To refer to an instance of something is simply a way of saying that you have one of something. For example (using the array again): After asking for a new array by coding "Array new: 20", you would have one instance of an array. If you executed the statement a second time you would have two instances of an array. OOP programmers like to use the term "instantiate" for creating an instance of an object.

Message

A message is how the OOP environment tells an object to do some work. Here is an example of a message using the array object. In the statement "myArray at: 2 put: 5", the object is myArray and the message is "at: 2 put: 5".

This message tells the array object to put the integer 5 at element 2. The message to myArray, "count := myArray at: 2", would tell myArray to return the value stored at element 2 and assign it to count.

Method, Instance Data, and Information Hiding

A method is the implementation of the code in the object that gets the work done when you send a message.

Example: The message "at: put:" that was used in the example above requires some code inside the array object to put the integer 5 at element 2. The code to do this element of work is called the method. The storage used to hold the array is contained in what is called "instance data". This refers to data used only by this instance of an array. The way a method is implemented is not important to the user of the object nor is the way the data is stored. This phenomenon is called "information hiding".

Refer to Figure 1 for a visual representation of the object, methods, messages and instance data.

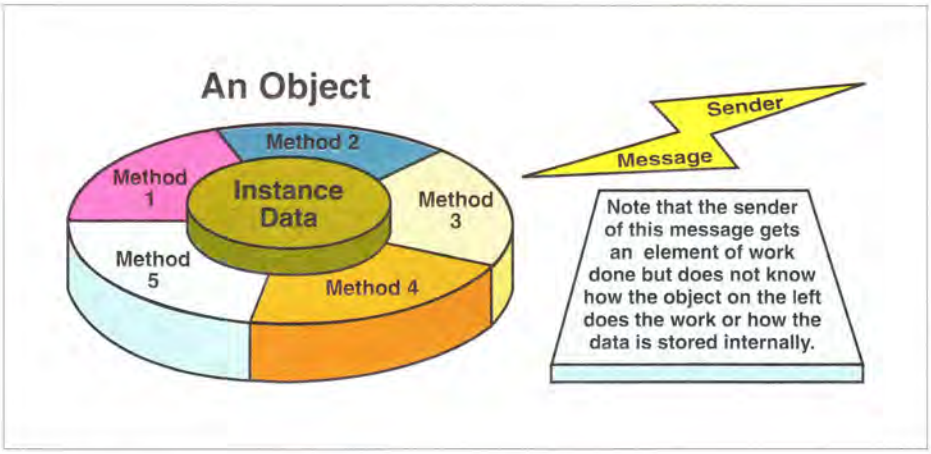


Figure 1. An Object

Inheritance

Inheritance allows you to create a new object that is similar to another one. The new object inherits the other object's behavior. In OOP, you inherit an object's behavior by creating a subclass of that object. This allows your object to inherit all of the behavior of its parent, plus you can add some function that is different from the original object. Example: You want to implement a stack. Because a stack is a specialized form of an array, one way to do this would be to create a stack object as a subclass of an array. Now you would add a push and pop function. The method for push would be to add the item to the array. The method for pop would be to take the last entry off the array and return it. By subclassing the array you would have a mechanism to easily store and retrieve data without needing to implement that mechanism yourself. This also is a good example of an object that could be added to the environment for reuse.

Polymorphism

This term probably scares off more potential OOP programmers than any other. It simply means that multiple classes of objects respond to the same message uniquely according to class. They all respond with the same expected results. A good example of this in Digitalk Smalltalk is the plus message. A float object and a fraction object would each return the sums of themselves and the number contained in the message as a response. Obviously the code inside the objects would be different to support the same message.

Garbage Collection

No, this has nothing to do with taking the garbage can out to the curb! Garbage collection means the OOP system frees up memory when it is no longer in use. This relieves the programmer of needing to be concerned about it. Example: In procedural languages a block of memory sometimes is passed from one process to another. The programmer has to make sure the memory is freed when the second program is done with it. In OOP, garbage collection frees the memory automatically when it is no longer referenced.

DEVELOPING AN OOP APP

Preliminary Design

Here is a popular style of developing an OOP application that works well for some programmers.

1) Write out a paragraph describing what your application is and what interactions it will have with the user.

2) Go through and highlight the nouns and the verbs in the paragraph. The nouns typically will relate to objects and the verbs to methods.

3) Make a first attempt at defining objects and methods. One way that works for some OOP programmers is the CRC (class, responsibility, collaborator) card method introduced by Beck and Cunningham[1]. The approach is to make a 3x5 card for each object and list the methods for that object on the card. Then all of the cards representing objects are grouped together, and a sample scenario of desired work is tested against the objects to see if they work well together. This is done by going step by step through the creation of each of the objects and then sending messages between objects to get the desired results . This determines whether the particular objects and methods chosen can handle the job. Throwing out objects and creating new ones is encouraged. The cards facilitate visualization of the messages passing between objects.

4) The OOP programmer now starts to implement the objects and methods. As new objects are created there is a natural desire in OOP to make the object more general so it can be added to the available classes of the environment. This will allow reuse of the object for the programmer and others.

Here is a sample application developed using this methodology. This program is a game called "Bible Book Jumble". It shows the books of the Bible arranged in Old/New Testament format. The user is challenged to place all the books in the proper order while

being timed. The user moves the location of a book by first selecting one book with the mouse pointer and then a second book. The two books swap position when the second one is selected. After each move the books are checked for proper order and the number of books in the correct place is displayed along with the elapsed time of the current game.

Refer to Figure 2 for a look at the game.

Per the above procedure, we first write out what the program is and what it will do, highlighting nouns (**bold**) and verbs (*italics*).

A **game** consists of a group of **books** *arranged* in alphabetic order that allows the user to *select* a book and *move* it to the proper location. The game also *keeps track* of **time**, *counts* the number of books in the proper place, and *displays* the status on a **score-sheet**.

Table 1 is a list of objects and methods using the noun verb method. Note that this first cut of objects and methods can be reworked considerably by the programmer as the program is developed.

Each of these objects now becomes a CRC card and the methods are written on the cards. An example of a test scenario would be to create a book and give it a name. This would be done

OBJECT	METHOD
BBJ Game	arrange, count
Book	Note that the verbs "select" and "move" expand into methods that allow detecting the mouse pointer, reading the name of the book, changing the name and color of the book and drawing the book. These methods are set/query name, is mouse in you?, set color and draw.
Time	set, add
Score-sheet	set text, display

Table 1

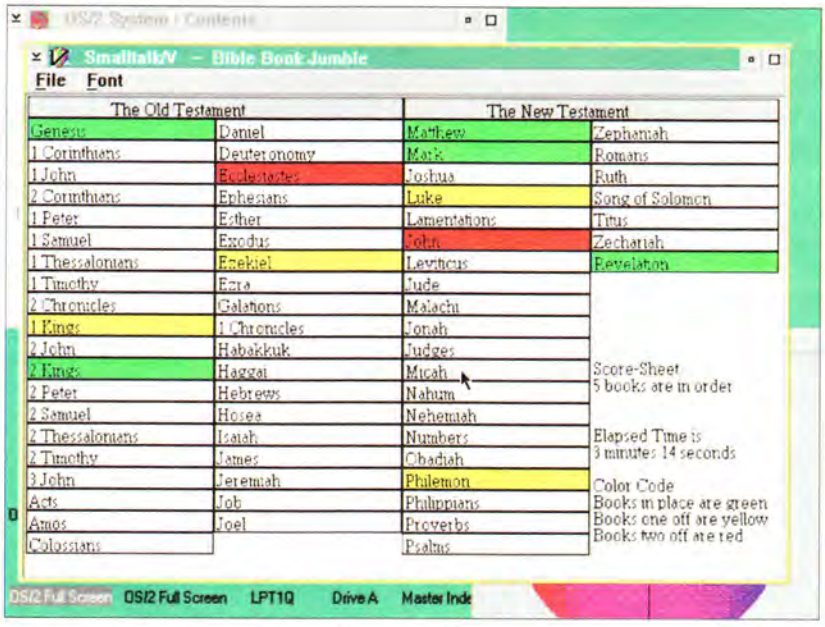


Figure 2. Bible Book Jumble Game

by the BBJ game object creating a new book object and then sending it a set name message and a draw message. One question that would arise is "What position would it occupy on the screen?". The result of this would be the realization that a set position method is needed for the book object, so this would be added. Additional scenarios would be acted out to perform the design of the game. A final set of objects and methods would finally be reached.

Final Design

In the final design, the BBJ Game object does the following:

- create an array object to hold the book objects.
- calculate the correct size and location for each book based on the window size of the BBJ Game.
- create one book object for each book.
- initialize each book object with its position, size, color, and name.

After this initialization phase, each book object has the ability to respond to the messages, draw, resize, change name, and change color. It also can respond to queries for its name and if the mouse pointer is within its boundaries.

Now, implementing the desired user function is just a matter of the game object asking a book object the right questions and telling it what to do. For example, after the user selects a book with the mouse pointer and releases button one, the BBJ Game object sends a "is mouse in you" message to each book object in the array of 66 books. Only one book will return true. The BBJ game object then sends a "set color" message to this one book. The book object will respond to this message by setting itself to the new color and redrawing itself to indicate to the user that it has been selected.

The time object already existed in Digitalk Smalltalk so I used it to set up the starting time and subtract the current time to determine elapsed time.

As mentioned before, the programmer is always thinking of ways to make objects more general so they can be added to the available classes of the environment. In this example we started thinking of the book object as just representing a book. We later realized it was really a just an area that could be used as a cell in a spreadsheet, a word in word game, or a letter block in a crossword game. So we created a generalized cell object. We then used it not only in the book object but also in the score-sheet object to represent a line of text.

The score-sheet object was implemented as an object that is a collection of the cell objects. These cell objects are used to display lines of text for instructions and scoring information.

Refer to Figure 3 for a block diagram of the objects involved.

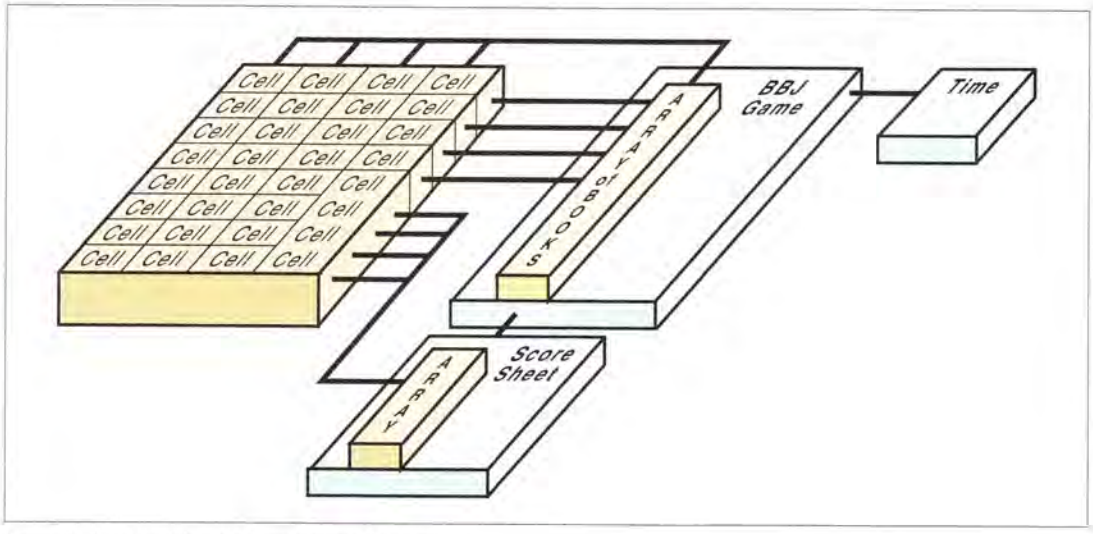


Figure 3. Bible Book Jumble Game Objects Diagram

POWERFUL PROBLEM-SOLVING CAPABILITIES

Here is a list of the fairly complex problems that were solved by using the OOP techniques.

- Subdivide the screen
- Correlation of the mouse to the screen
- Enable swapping of information from one place to another
- Tracking the location of each book against the proper location
- Keep track of time
- Sort the books

Following are some of the powerful Digitalk Smalltalk concepts used to solve these problems. These examples are given to show a sample of the powerful function available to the programmer in the Digitalk Smalltalk environment.

- asSortedCollection (to sort the books)

This is a transformation that is enabled by just adding this one message to the end of an array statement. In this case it allows the programmer the ability to easily sort an array. Sorting is usually much more complex in other programming languages. There are many powerful transformation messages in Digitalk Smalltalk.

- Time object (to keep track of time)

This allows querying time in many formats and enables math operations on time data. This time object gives the programmer the ability to work with time without worrying about doing conversions that usually are associated with this subject.

- Select (for scoring and correlation)

Digitalk Smalltalk has some powerful one-liner statements that perform operations against an entire array without do-loops. "Select" allows the programmer to ask an entire array if any element in it evaluates true to a test expression. This is a concept that may be hard for the beginner to understand. We mention it here because it is important to know that such powerful statements exist if a programmer is willing to try them.

Since "select" to track the location of the books in relation to their correct position was so easy, We decided to use colors as hints. All books in the correct place are green, one off in either direction are yellow and two off are red.

- Pen (for graphics)

This was used for drawing boxes, filling with color, and drawing characters. The pen metaphor is easy to use and powerful for graphical work. In OS/2 Presentation Manager®, the programmer is required to make a series of complicated graphic function calls to set up even a fairly simple graphic drawing. In Digitalk Smalltalk this is done for you. The Smalltalk programmer creates a graphic pane and uses the pen.

SMALLTALK/V PM IMPLEMENTATION ON OS/2

Programmers that are used to editing source, and then dumping it to a compiler and waiting three minutes or more for compile and link, will notice a difference with Digitalk Smalltalk/V PM right away. In Smalltalk/V PM you create an object and then work on one method at a time. A method is typically 1 to 10 lines of code. As you save each method the clock icon pops on for 1 to 2 seconds and your code has been compiled. This compilation happens so quickly it seems more like just a syntax check (which also take place at the same time). There is no environment setup, no makefile and no include files. Refer to Figure 4 to see what the class browser looks like. This is where you edit your objects and methods. In this example the Cell object "pointInYou" method is shown. This method returns true if the point passed is within the cell or false if it is not (*See Figure 4*).

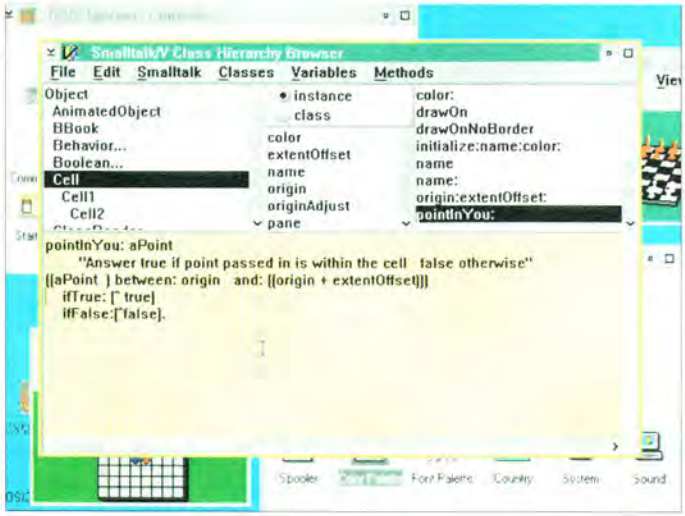


Figure 4. Smalltalk/V PM Class Browser

When the code compiles, it goes into an executable file called V.EXE which continues to grow as you add more methods. When you are finished adding functions, you rename V.EXE to your program name and you are done. You can deliver this EXE along with the Digitalk Smalltalk/V base DLLs as a runtime application. If you have a Digitalk development license you may distribute this runtime application without paying any royalties to Digitalk Inc. This is stated in the Digitalk Smalltalk/V PM Developers Guide[2]. The development license also gives you access to the development VPM.EXE and DLLs (which may not be distributed).

You also have the option of putting part or all of your application into OS/2 DLLs. This is done by using the Object Library Builder. The Digitalk Smalltalk/V PM Developers Guide suggests that this could be used to organize into modules that are individually replaceable or to share sub-components across multiple applications.

SMALLTALK VS. 'C'

Veterans of 'C' / Presentation Manager will find some advantages in using Digitalk Smalltalk/V PM.

- Use of a rich class library with powerful functions.
- No need to worry about allocation and freeing memory.
- No need to define structures and control blocks.
- You can create a PM application without dealing with the complexity of PM programming.
- Immediate compilation.
- No need for complicated makefiles.
- No need for complex include environment setups.
- Quick code changes. For changes other than those that are executed on initialize, you can actually change the code and test it without closing the program. This encourages prototyping and experimentation.
- Built in debug environment included with the development environment.

- Walkback instead of protection fault. Digitalk Smalltalk brings up a "walkback" debugger when an object can't process correctly. This allows stepping through the code up to the point of error. This is much more programmer-friendly than the protection exception TRAP D screen that C programmers are used to.
- Portability. If you develop an application on Digitalk Smalltalk/V PM, it can be run on Smalltalk/V Windows with minor modification.

ADDED FUNCTION IN THE LATEST SMALLTALK/V PM RELEASES

Digitalk Inc. is continually updating Digitalk Smalltalk/V PM to keep up with OS/2 releases. Here is a list of enhancements by release. We have included this list to point out that Smalltalk/V PM is staying current with the latest OS/2 enhancements.

Version 1.3 enhancements

- Runs on OS/2 2.0
- CUA Controls
  - File Selection Dialog
  - Notebook
  - Slider
  - Container
  - Value Set
- National Language Support (NLS) Double Byte Character Support (DBCS)
- Help manager support
- Performance Profiler
- Window Management compatibility with Digitalk Smalltalk/V Windows

Version 2.0 and Future Design Direction

- New architecture to take advantage of 32-bit architecture at run-time
- Full exploitation of OS/2 multitasking by using OS/2 threads

SUMMARY

The author's own experience with learning and using Digitalk Smalltalk/V PM can be summed up as follows:

- It is a very useful tool for learning OOP principles in general. These principles were very useful in understanding the OS/2 System Object Model (SOM), which is used to create OS/2 Workplace Shell integrated applications (also using object-oriented technology). It also would be a useful prerequisite to learning C++.
- Move projects over to OOP slowly. It takes a while to learn to use it effectively, although it is worth the effort.
- Consider the performance aspects of the target application early in the design cycle. OOP technology tends to produce larger executables. This can have a negative impact on memory usage and response time. This certainly does not rule out OOP environments as application development tools. Just make sure you understand up front both the environment and the target machine the application is going to run on.

REFERENCES

1. *Cunningham, W. and Beck, K.: "A Laboratory For Teaching Object-Oriented Thinking", in Proceedings of OOPSLA-89, October 1989.*

2. *Smalltalk/V PM Developers Guide, Digitalk Inc.*

ACKNOWLEDGEMENT

Many of the concepts and techniques mentioned in this article are taught in "Object-Oriented Programming and Design with Smalltalk", a course offered by IBM Systems Research Education Center, Thornwood, NY.

Gerald C. Strope, *IBM Programming Systems Division, Westlake Laboratory, 5 West Kirkwood Blvd., Roanoke, Texas 76299.* *Mr Strope is a senior programmer working on a PRGS development project. He joined IBM in 1965 at the Aurora, Illinois branch office. His career has spanned many aspects of computing, including teleprocessing, hardware service planning, and hardware service cost estimating. In 1984, he became involved in software by working on the development of a new IBM internal service cost estimating program. He joined the application development division in the Dallas area in 1986, where he has held lead development and design positions.*

Software Tools

SegMentor™: Minimizing Swapping and Linking Overhead

Jon Holliday

Chris Suver

To achieve optimum performance, functions must be grouped within pages or segments

by Jon Holliday and Chris Suver

Mention segment map tuning to a programmer today and you'll usually get a loud groan. Most of us detest such an assignment, knowing it to be a trial and error process that is tedious, time consuming, and very frustrating. Unfortunately, it is a job that sometimes can't be ignored, unless you don't mind a program that "thrashes" in low memory situations — using nearly all the computer's cycles on retrieving and linking segments, and relatively few on actually executing functions. And even with plenty of memory, an untuned segment map can cause from 5% to 20% overall performance degradation, due to the overhead of far calls. This is especially true on larger and function intensive (like C++) applications. Fortunately, there is now a tool available from MicroQuill called the SegMentor™ that allows this task to be automated.

DYNAMIC LINKERS AND SEGMENTATION

Through dynamic linking under OS/2® and Microsoft Windows®, applications that are larger than available memory can be run by having the memory manager load code segments in from disk as needed at runtime. Because these code segments are dynamically linked into the running program, they can be loaded anywhere in memory. So the developer no longer needs to decide what combinations of code segments can be resident at the same time, as is the case when breaking a program up through the use of overlays. But while dynamic linking relieves the programmer of having to decide where code segments will reside, it does not address the issue of optimizing the content of each code segment.

The content of an application's segments (the way in which its functions are grouped into segments) can affect its performance. Particularly in low memory situations, the effect on performance is significant. So, if the developer is concerned about performance, he is still left with resolving the allocation of functions to segments.

HOW SEGMENTATION AFFECTS PERFORMANCE

Good segmentation groups functions into segments based on how frequently the functions call each other. The goal is to minimize the number of cross-segment function calls that occur as the application runs. Overall application performance is improved in two ways by minimizing cross-segment calls.

First, when a cross-segment call occurs, there is some probability (which is a function of the amount of available memory) that the segment containing the target function won't be in memory. If it is not, it will have to be loaded from disk, which is a very expensive operation. A call to a function on disk will take 100 to 1000 times longer than a call to a memory-resident function. For a given application, there's a minimum amount of memory (which depends on the operation being performed) that is required for the application to perform acceptably. When memory drops below this level, the application will still run, but it will begin disk thrashing — a high percentage of its function calls (or even function returns) will involve loading a code segment from disk. In these cases, the function calls and their associated disk hits often account for 85% or more of the

total time required for an application to execute. Good segmentation can have a dramatic effect in these situations, often producing overall performance improvements of 5X and more.

In addition, a well-segmented application will reduce its minimum memory requirement. A rule of thumb is that a randomly or poorly segmented application requires N! times as much minimum-memory as it does when it's well-segmented.

A second reason for minimizing cross-segment calls is that cross-segment calls are always far calls under DOS, Windows and OS/2 1.x. Far calls take longer to execute than calls within the same segment. A far call within the same segment can be optimized by the linker using /FARCALLTRANSLATION option so that it takes less than half as long to execute. And a near call takes even less time to execute, as shown in Figure 1. So, if an entirely memory resident application spends 10% of its time just making function calls, good segmentation can cut this time in half, thus speeding up the application's overall performance by 5 percent.

The near/far call distinction goes away with the full 32-bit linear addressing schemes in OS/2 2.0. However, even in a flat memory model, applications will continue to run into low memory situations, forcing 4K pages to be loaded from disk. Thus, paged environments will still require functions to be grouped intelligently within pages for optimum performance.

An Unrealistic but Illustrative Example

Imagine an application has 5 functions: a, b1, b2, c1, c2. Each function is about 2K bytes in size, and you want to segment the application into 4K code segments (so two functions will fit in each segment, and the application will have to be split into three segments). In a typical runtime usage of the application, a calls b1 10 times, then a calls c1 10 times. Each time b1 is called, it calls b2 10 times, and each time c1 is called, it calls c2 10 times. So a total of 220 function calls occur when the application runs.

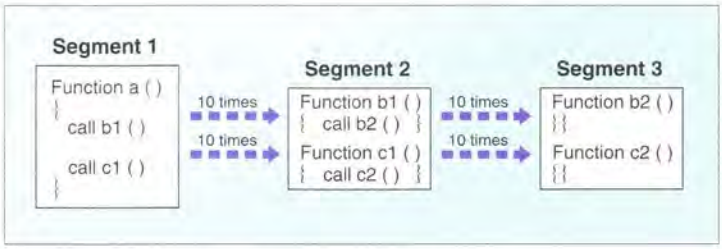


Figure 2. Worst Case Segmentation Example: 220 Far Calls

A worst case segmentation for this application would be to place b1 and c1 together in the same segment, b2 and c2 together, and a in a segment by itself, as in Figure 2. In this case, all 220 function calls that occur will be cross-segment calls rather than in-segment calls.

Processor	Call	Protect Mode Clock Cycles
80286	Far	26
	Far using /FARCALLTRANSLATION	13
	Near	7
80386	Far	34
	Far using /FARCALLTRANSLATION	12
	Near	7

Figure 1. Clock Cycles for Near and Far Calls
Note: Information from Microsoft C Advanced Programming Techniques.
© Microsoft Corporation, 1990.

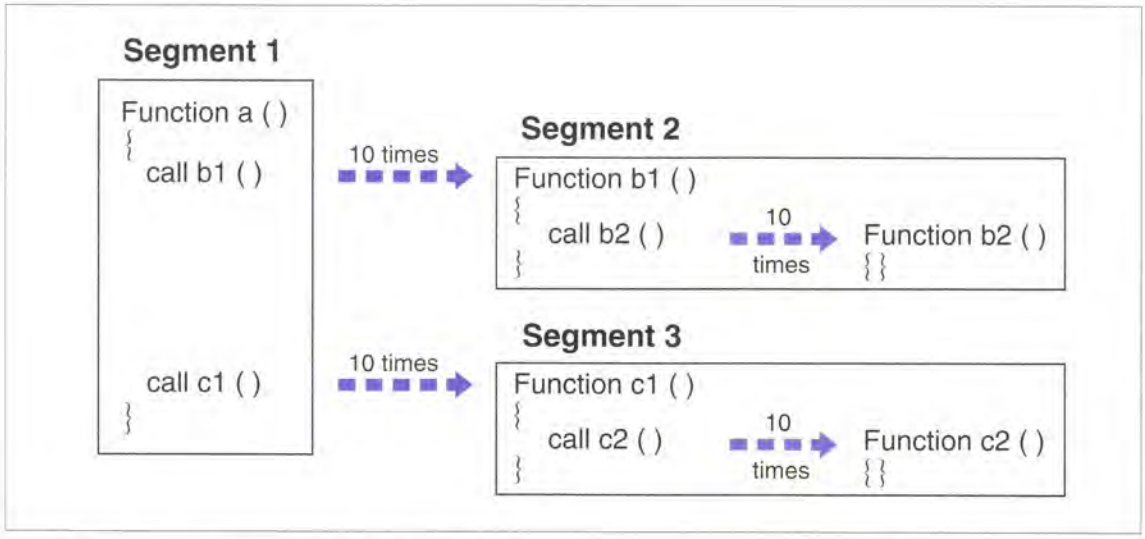


Figure 3. Best Case Segmentation Example: 20 Far Calls

The best case segmentation for this application is to put b1 and b2 together, c1 and c2 together, and a by itself, as in Figure 3. In this case, there are 20 cross-segment function calls and 200 in-segment function calls. If there is at least 12K of memory available on the machine (and no other applications are running), all of the code segments will fit in memory and 0 'disk-hit' function calls will occur for both the worst case and the best case segmentation. The worst-case application would run a little slower though, because all of its function calls are far calls.

If only 8K of memory were available (so that only 2 code-segments could be resident at a time), the worst case solution would induce 20 disk-hits while the best case solution would only induce 2. And if only 4K were available, the worst case solution would induce 440 disk hits (returns from functions in addition to calls can require a code segment to be read from disk), while the best case solution would only induce 40.

Ramifications of Segment Size

A possible solution to the segmentation problem would be to have just one function per segment. This would allow an application to run efficiently in the minimum amount of memory. The application would make the best possible use of memory and would have the fewest possible disk-hit calls. But there are two problems with this approach.

The principal problem is that Windows and OS/2 both place a limit on the number of segments allowed in an application. This is especially a problem with Windows where an application is limited to only 255 segments. So this solution isn't feasible for larger applications. And those applications that are small enough to use this scheme probably don't need to be very concerned about low-memory/segmentation issues anyway.

The second problem is that assigning one function to a segment forces all calls to be far. For many applications, this won't have a very big impact on performance, but for some applications (either because they're very time-critical or very function-call intensive) this can be important.

Using very large segments is another alternative. Larger segments have the advantage that they minimize the number of far calls that occur. However, as segments grow larger, more time is required to read and dynamically link them into a program. Also, as the segment size increases, the minimum memory required by the application tends to increase.

In practice, a code segment size of 4K to 8K produces the best overall results. It is not by chance that paged memory schemes have settled on 4K as the fixed page size.

Summary of the Segmentation Problem

The problem, then, is to place an application's functions into segments (of about 4K to 8K apiece) in a way that will minimize the number of cross-segment calls that occur when the application runs. The data needed to solve this problem are: a list of the functions in the application, the size of each function and finally, the sequence of function calls that occur when the application runs (in a typical session).

While the problem of finding an optimal solution is a simple one, the solution space that must be searched is very large. For example, assuming you have to place n equal sized functions into m segments, there are on the order of m*n possible solutions to consider. And, at least to our knowledge, there are no algorithms available that substantially trim this solution space.

INADEQUATE SOLUTIONS

Leaving segmentation up to the linker is unacceptable because linkers don't have access to the runtime function call history of an application. To group functions into segments, a linker can make use of two pieces of data. First, it can group functions based on how they are grouped in their source code files. This is a reasonable idea, since functions in the same file tend to call each other. Second, a linker can make use of the 'static' call-tree data on an application. A linker can know, for example, that function b1 calls function b2. What a linker can't know is how many times b1 calls b2, and it also can't know about hidden function calls that occur through function pointers, call backs, etc.

Generally, even making use of the above data, a linker's segmentation of code is only slightly better than random segmentation. This is because the 'static' data that is available to the linker is simply inadequate.

Another option is to segment an application by hand. This has three drawbacks. First, the programmer may have the same problem the linker does: lack of sufficient data. In particular, most DOS/Windows/OS/2 profilers do not provide information on 'who calls whom, and how many times'. The runtime function call history is critical information for segmentation decisions. Thus, a programmer must make educated guesses about how strongly linked a given set of functions are (that is, how frequently they call each other). For small applications with relatively few functions, this can be a manageable method. But with larger apps, the possible combinations get too numerous to consider.

Second, even given the necessary data, finding a good solution by hand is very difficult. What the developer typically finds is that performance is improved in the area he had considered, but at the expense of slowing other parts of the program. This is because the reallocated functions also were used in some other part of the program that was not considered. The change removed these functions from another segment where they were frequently used, thus slowing that area of the program.

When doing this job by hand it is impossible to know every implication of a change in segmentation. Very simply, computers can do a better job of solving problems of this class than people can. Given a reasonably sized application, a person simply cannot consider all the ramifications of a single solution, let alone compare that solution to others.

Finally, hand solutions are labor intensive, which, of course, is very expensive. This is especially the case when an application has to be changed fairly frequently, thus requiring frequent changes to its segmentation.

SegMentor — AN AUTOMATED SOLUTION TO THE SEGMENTATION PROBLEM

Well, there is a moral (or happy ending, or something like that) to this story, which is that a rather straightforward, automated solution to this problem now exists.

SegMentor, by MicroQuill, is a product which specifically addresses the problem of application segmentation.

It goes right to the heart of the problem (the lack of a runtime function call history for the application) by capturing this data through some clever tricks with the chkstk routine. Then it runs the data through a series of algorithms (much like those for auto-routing printed circuit boards) to compute a (nearly) optimal segmentation layout. This layout is then provided in a format that's easily incorporated back into the application (for C, this is done through a .h file). At present, SegMentor supports C (Microsoft and Metaware) and C++ (Glockenspiel and Zortech) for the DOS, OS/2 and Windows platforms.

The remainder of this article describes in more detail how SegMentor works.

Capturing the Critical Data

The first step in using SegMentor is to build a database on the application (.exe or .dll) that's being segmented. The basic information needed is: a list of functions in the application, their sizes, and the runtime calling relationships among the functions. But in order to collect the runtime calling data (and in order to make some other things work) some secondary data items are also needed (for example, the name of each function's default segment and source code file). Data are drawn from three sources: the application's .map file, the application's source code, and the 'log files' that are generated for the application (which contain the application's runtime function history).

.map file data: The .map file provides the name of each function in the application together with its (executable) size. It also provides the name of each function's default segment (this is used in tracing the application's function activity).

Unfortunately, functions declared as static don't show up in an application's .map file. To get around this problem, a SegMentor utility called Fixup is used to change the static keyword (when used in reference to functions) to a macro, such as LOCAL. Then, by defining this macro as 'nothing', it is possible to compile without static functions and thus generate a full .map file (for other compiles, the macro can be defined as 'static').

source code (.c file) data: The application's source code points out those functions in the application's .map file for which there is no corresponding source code. This may be because they are from the standard library, or from third-party libraries, or because they're written in assembler. In any case, without source code there's no mechanism (in Microsoft C) for assigning a function to a particular segment, so such functions are ignored by SegMentor, and they wind up in default segments based on how they're grouped into object files. (Note: assembly code can be segmented, but not as easily as C can be.)

The source code also provides a prototype for each function, and the name of the source file that each function is found in (these items are used in conjunction with the alloc_text pragma when incorporating a segmentation scheme back into the application). In addition, the source code is used in determining whether a function is far or near, and whether it uses the Pascal or C calling convention (these items are used in tracing the application).

Last, the source provides 'static' call-tree information: all the non-hidden function calls that can be derived from the source code (though frequency information is not available). This is used to supplement the runtime function call history: if a given function doesn't appear in the run-time history because the developer fails to exercise it for some reason, the static function-link information can then be used to make reasonable segment-placement decisions for that function (these static links are generally assumed to have a frequency of 1).

runtime function call (.log file) data: To get a run-time function call history for an application, a 'traceable' version of the application must be built. First, the application is compiled with stack checking enabled. This causes a call to the chkstk function to be placed at the beginning of each function in the application. Then a special trace.lib is built using the application's database and a SegMentor utility. trace.lib contains a special version of chkstk which logs function calls and returns to a disk file (in addition to performing chkstk's other duties).

How does MicroQuill's chkstk work? Each time it is called, it determines which application function called it (and in that way, which application function is currently being called) by examining the return address on the stack which corresponds to the call to chkstk. The return address is correlated with information from the application's database (which is coded into trace.lib) in order to figure out which function made the call. Also, in order to trace function returns, chkstk saves the return address that corresponds to the call to the current application function, and replaces it with a call to a MicroQuill routine called rtn_hook. That way, each time an application function does a return, rtn_hook gets invoked. When rtn_hook is called, it logs the function return to disk and then does a jump to the saved return address.

Once a 'traceable' version of the application has been built, it can be thoroughly exercised in order to generate a log file of runtime function call activity. Ideally, each part of the application should be exercised. Furthermore, it's best if each part of the application is exercised in proportion to the percent of time that it's used. So if end users spend 75% of their time using option X, then realistically 75% of the log file data should relate to option X (even if option X only represents 10% of the application's code).

One option available with SegMentor is to create separate log files for the different parts of an application. Then, when a log file is incorporated into the database, it can be weighted more heavily than others by having its data multiplied by some value. For example, you might have the log file data for option X weighted 7 times more heavily than it normally would be. (Another option would be to create a single log file, and then just spend lots of time exercising option X. But in practice, this latter option can be awkward, as it might dramatically increase the amount of logfile data needed for a good segmentation).

For a large application, the log file(s) produced will typically be several megabytes in size (3! bytes are needed to store each function call and each function return).

Computing a New Segmentation Layout

Using SegMentor utilities, the above data are gathered into a database file. Then a utility called Segment can be run, which searches for an optimal segmentation for the application. Initially, Segment is passed several parameters, including the target segment size that's desired (4K is recommended), and the log database. Segment will crank away searching for solutions for as long as desired; whenever it finds a solution that's better than all previous solutions, it saves it. For a large application, Segment usually finds a good segmentation in one or two hours. To get the best possible results from it, it can be left to run overnight.

Segment rates a given segmentation scheme by adding up the cross-segment function calls that the scheme produces. The fewer the cross-segment calls, the better. So in our previous example, the best case segmentation would be given a rating of 20, while the worst case segmentation would be given a rating of 220. This rating function is optimum for reducing far calls. It also finds segmentations that produce a minimum number of 'disk-hit'

calls when the application is run in a low-memory environment. Note that the most accurate rating function for minimizing disk-hit calls would be to simulate the sequence of calls that occur as the application runs, by assuming a given amount of available memory and an LRU swapping algorithm. However, such a rating function is impractical because it's so expensive to compute, and it turns out (not surprisingly) that minimizing the cross-segment-call rating happens to do a very good job of minimizing the disk-hit rating.

Segment first makes an initial placement of functions into segments. Using a mini-max algorithm, it's able to make an initial placement that's fairly good. This placement is random in the sense that each initial placement it makes tends to be different. Once the initial placement is made, Segment begins a phase in which it randomly moves functions around from one segment to another in an effort to improve the segmentation's rating. In this phase, Segment can either do 'iterative improvement' or 'annealing'.

In iterative improvement, only moves which lower (and thereby improve) the segmentation's rating are accepted. Segment accepts 'downhill' moves (that is, moves which lower the segmentation's rating value) until no more no downhill moves can be found. At that point, it makes a new initial placement and repeats the process. (Whenever it finds an improved segmentation scheme, it saves it to the database). With annealing, Segment will initially accept uphill moves (moves which raise the segmentation's rating value). For example, immediately after making the initial placement, Segment might accept a move which raises the rating by 50. Over time, the size of the uphill move that Segment accepts is reduced, until it finally reaches 0, at which point the process becomes one of iterative improvement.

If you imagine the solution space to be a 2 dimensional (hilly) plane, in which altitude corresponds to the segmentation's rating value, then with iterative improvement, Segment can only drop to the bottom of the closest valley or hole that the initial placement puts it in. With annealing, Segment can climb out of the hole that it's initially placed in, which allows it to 'explore' neighboring holes in addition to the one it's initially placed in. This gives it the possibility of finding deeper holes.

Both techniques are worthwhile. Given enough time (24+ computation hours) annealing seems to produce slightly better results.

Integrating the New Layout Back into the Application

Once Segment has been run on an application's database, all that's left to do is to integrate the resulting segmentation scheme back into the application. With C, the mechanism that's used to 'enforce' the segmentation is the alloc_text pragma, which allows one to place a function in a particular segment.

To do this with SegMentor, the utility Makeinc is run. Makeinc reads the database and outputs a single include file called project.h. This include file contains all of the necessary alloc_text pragmas for the application. As part of the preparation for using SegMentor, a #include of project.h must be added to each of the .c files that make up the application (the Fixup utility can add this at the same time it 'removes' static functions from the application).

With project.h built, the application is re-compiled and re-linked (this time without stack-checking and without the trace.lib file) resulting in a well-segmented application.

SUMMARY

To achieve optimum performance in an application, functions must be grouped within pages or segments, respectively, such that calls from one page/segment to another page/segment are minimized. This is true for both 32-bit flat memory models and 16-bit semented memory models. This grouping also will minimize the amount of disk swapping of an application's code and help prevent an

application from thrashing. With SegMentor, the process is mostly automated and saves developers from having to spend days trying to group their functions optimally by hand. Such efforts, even once complete, still would be affected by any change in the application and, then, require the developer to go through the process of grouping the functions together all over again. The relatively small amount of time and effort required to segment an application using SegMentor, coupled with the fact that it can run unattended, makes it useful as an integral part of the development process.

Jon E. Holliday, *2Bg/671/TB3, IBM Programming Systems Laboratory, 11000 Regency Parkway, Cary, NC 27512-9968. Mr. Holliday is a senior associate Programmer in PM Controls Development. He joined IBM in 1987 and worked on the OS/2 1.2 Dialog Manager. Mr. Holliday is currently working on 32-bit PM Controls for OS/2 2.0, and "thunk" code to allow 16-bit and 32-bit programs to communicate. He received a BS in Computer Science from North Carolina State University.*

Chris Suver, *MicroQuill Inc., 4900 25th Avenue N.E. #206, Seattle, WA 98105. Mr. Suver is currently founder and president of MicroQuill. The firm specializes in CASE tools for performance analysis and tuning. MicroQuill sells SegMentor and Performance Tracer to commercial software firms and organizations developing large internal projects. Mr. Suver began developing commercial products in 1974, and has founded a timesharing service as well as Precedent Systems, the current market leader in office management solutions for physical therapy practices. His career has included stints at Microrim, Generic Software, and CADDEX Corp., where he led teams developing database engines and languages, and desktop publishing software.*

Multimedia and Graphics

Introduction to IBM Multimedia Presentation Manager/2™

Brad Noe

Bill Lawton

MMPM/2 Provides a single consistent Multimedia Programming Interface

by Brad Noe and Bill Lawton

This article discusses the features of Multimedia Presentation Manager/2 (MMPM/2™). This new extension to OS/2 2.0 provides a single consistent multimedia programming interface for a wide range of existing and future multimedia devices.

At Fall COMDEX 1991, IBM announced an array of multimedia hardware and software products and a statement of direction for multimedia Presentation Manager/2. MMPM/2, an extension to OS/2 2.0, provides an application programming interface as well as a run-time environment for multimedia applications. These programming interfaces are consistent with those in Microsoft Multimedia Windows Extensions 1.0 for DOS Windows 3.0, to better enable cross platform application development and migration. MMPM/2 together with OS/2 2.0 and various multimedia hardware products enable applications to use sound and video through the use of an easy-to-program interface. MMPM/2 provides three subsystems for multimedia application development: a Media Control Interface (MCI), Synchronization/Streaming Programming Interface (SPI), and the Multimedia I/O Programming Interface (MMIO).

There also is a developer's toolkit available for MMPM/2 to facilitate the integration of multimedia into new or existing applications. The MMPM/2 Toolkit includes C language bindings, libraries, and C sample code for application developers. The toolkit also includes sample stream handlers (more on this later) as well as a sample audio device driver.

In addition, MMPM/2 includes a generic media player "applet" to play various waveform audio files, MIDI files and CDs.

The first version of MMPM/2 will consist of software support for the IBM M-Audio Capture and Playback Adapter, waveform audio, MIDI, and the Pioneer 8000 videodisc. Digital and analog video interfaces will be provided in future releases of MMPM/2 (*See Figure 1*).

MEDIA CONTROL INTERFACE

The Media Control Interface for MMPM/2 provides a generalized interface to control multimedia devices. Multimedia applications deal with "logical devices", sending commands to devices through either a procedural interface or a string based command interface. Some of the devices that applications may deal with are: CD audio, waveform audio, MIDI, audio amp/mixer, and videodisc. The commands available for each device vary with the nature of the device, however, functions that are common across devices are invoked in a consistent manner. The definition of this interface was jointly developed by IBM and Microsoft. This interface consists of two main APIs and a set of multimedia commands. The two APIs are MCISendCommand and MCISendString. These two APIs allow applications to use multimedia devices in a device independent manner. The difference between the two APIs is that MCISendString allows the application to send in a textual string representing the command while MCISendCommand is a procedural interface. Examples of commonly used commands are OPEN, CLOSE, PLAY, RECORD, STOP, and SEEK. Before an

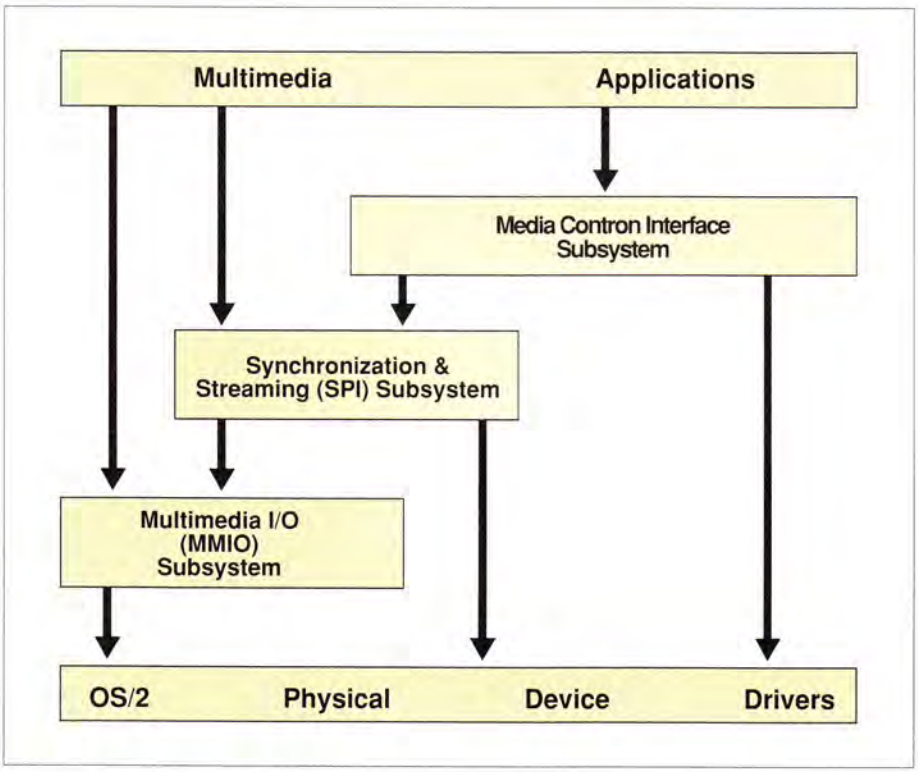


Figure 1. MMPM/2 System Diagram

application can use a device, it must first create a device context by opening the device. When a device is opened, a device ID is returned to the application. This device ID serves as a unique "handle" to the device context, to identify the device context on subsequent messages to the device.

The MCISendString API allows textual command strings to be sent to MMPM/2, which has a parser and command tables to convert the textual string into the MCISendCommand equivalent. An example of strings that might be used are:

- `open cdaudio alias mycd`
- `play mycd`
- `close mycd`

This example plays an entire audio compact disc (CD). Notice the use of the word alias.

An alias is a name that can be used to represent an opened device context. Aliases can only be used through the string interface.

One of the advantages of the MMPM/2 Media Control Interface is its device virtualization. Rather than requiring each application to be responsible for its own device state management, when multiple applications are sharing devices MMPM/2 maintains the state of devices on a per context basis. This allows two or more multimedia applications to share the same device, either serially or in parallel, depending on the operation or limits of the device. For example, two applications can open a CD device and share the device serially.

```
application A - "open cdaudio alias cdA shareable"
                "play cdA"
application B - "open cdaudio alias cdB"
                "play cdB"
```

Application A opens a device context of the CD device as shareable and then proceeds to play the device. When application B opens a device context of the same CD device, application A would receive a PM message MM.MCIPASSDEVICE with the low word of MSG2 equal to MCI.LOSING.USE. This message tells application A that its device context is being made inactive and its current state is being saved. The CD device would then stop playing, since the device context is inactive. Application B would receive a PM message MM_MCIPASSDEVICE with the low word of MSG2 equal to MCI_Gaining_Use. This PM message (MM_MCIPASSDEVICE) indicates to application B that its device context is active (it now owns the device). Application B can then proceed to play the device. Two commands, ACQUIRE and RELEASE, are available to applications to make inactive device contexts active again.

SYNCHRONIZATION/ STREAMING INTERFACE

Multimedia software in general typically has the requirement of providing continuous data flow from a storage device to an input/output device. For example, playing a waveform file from a hardfile to an audio adapter in a personal computer must be done without interruptions to the data flow. Providing this continuous data flow for MMPM/2 is the job of the Synchronization/Streaming programming subsystem (*See Figure 3*).

The SPI subsystem consists of a low-level programming interface that can be used by applications or other subsystems within MMPM/2 to stream data from a source device to a target device. Figure 3 depicts how an MCI driver invokes the services of the Synchronizer/Stream manager to transport data between devices. Another requirement

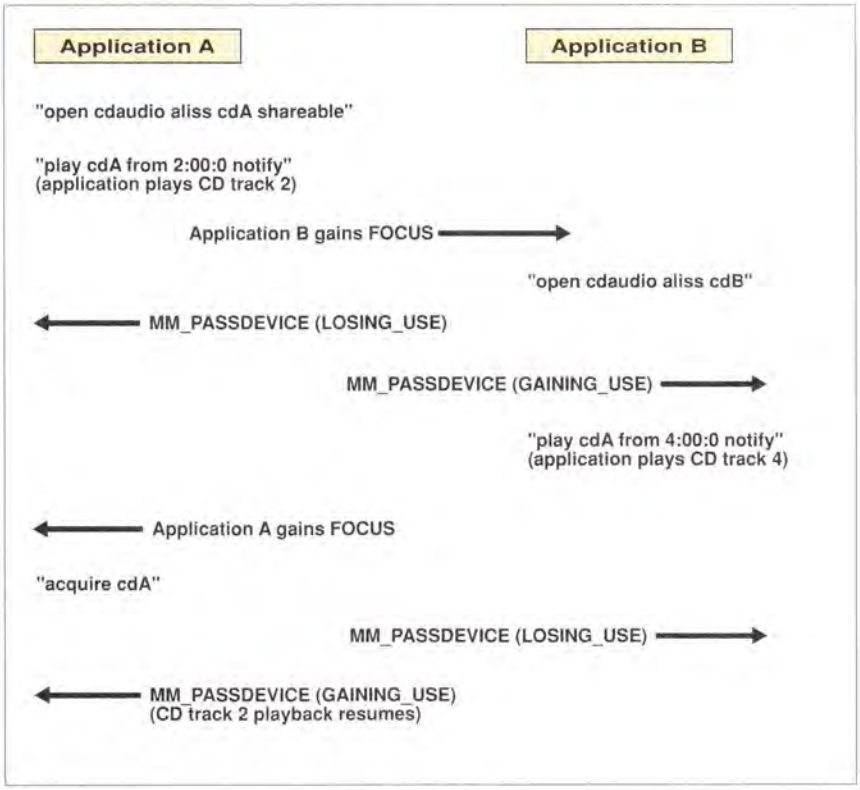


Figure 2. MMPM/2 Resource Management

for multimedia applications is the ability to synchronize multiple data streams, and provides notification of real-time events. MMPM/2 provides both of these functions through the use of the SPI subsystem.

The SPI programming interface is a generalized mechanism for data streaming and is independent of the type of data being streamed. The actual subsystem consists of a central stream manager with peripheral stream handlers. Each of these stream handlers is usually specific to a type of input/output device. For example, there is an audio stream handler which interfaces to audio device drives using an audio device driver protocol. There is a file system stream handler, which interfaces to the OS/2 file system to read data from an existing file or to write data to a new file. All stream handlers act as either a source or a target for a stream and deal with either filling the stream with data or taking data from the stream.

There are also "filter" stream handlers like the MIDI stream handler, which receive data from one stream, transform it and put it into another stream to be received by another stream handler. This particular stream handler acts as both a source and a target within the same playback instance. Typically, a given stream handler can be a source in one instance and a target in another. For example, the audio stream handler is a target stream handler during a playback of audio data and it is a source stream handler while recording from the audio device to the hardfile.

MMPM/2 is supplied with the basic stream handlers needed to stream MIDI or waveform data to and from a multitude of devices. The system is extensible by adding new stream handlers to the system. In the future, new source stream handlers, like a video stream handler, can be added to the streaming subsystem without having to change the target end of the system.

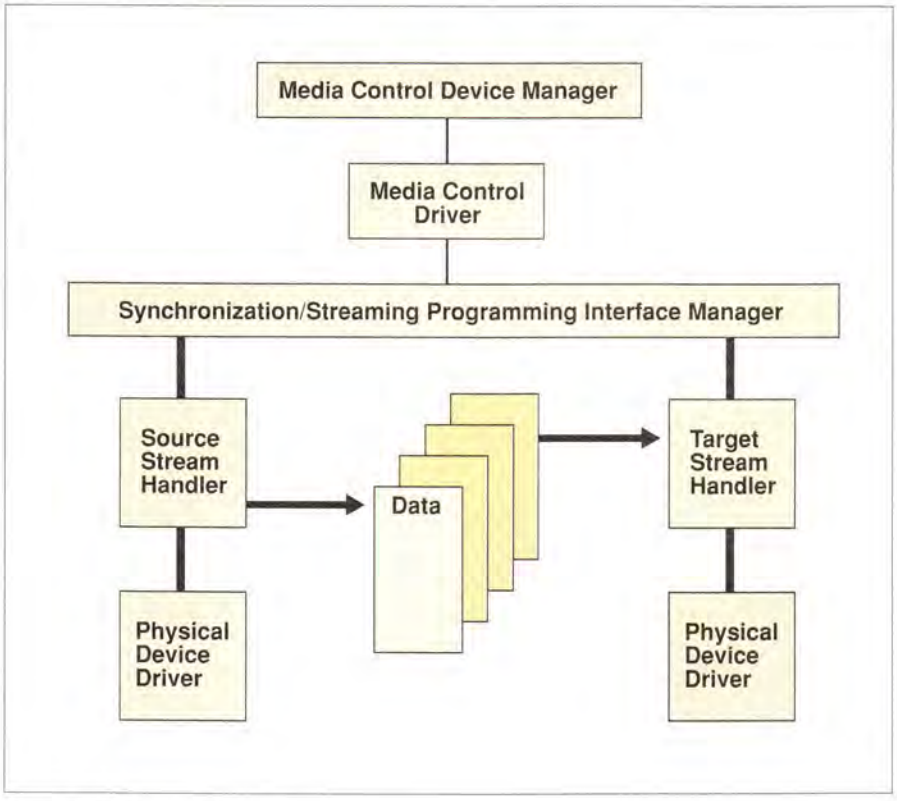


Figure 3. SPI System Diagram

The SPI interface provides the ability to stream data between devices as well as synchronize the output devices in a generic fashion. It accomplishes these two tasks through a series of commands to the Sync/Stream Manager. There are commands to create, start, stop, and seek a data stream. These mostly deal with the data transportation function of SPI. The synchronization commands allow an application to create synchronization groups—a collection of data stream with one data stream acting as the master and the others acting as the slaves. A master stream provides synchronization "pulses" to the slaves on regular predefined intervals so that they can remain synchronized to the master stream.

MULTIMEDIA I/O PROGRAMMING INTERFACE

The Multimedia I/O (MMIO) subsystem is an extension to the OS/2 file services, providing a powerful mechanism for accessing and manipulating multimedia data files. These files contain a variety of media elements such as digital audio, digital video, images, graphics, and others. Along with the diverse data elements, the organization of these files is often complex due to the disparate types of information they contain and the number of different file formats. MMIO is a standardized method for applications to refer to these files, read and write data to the file and query about the contents of the file. This standardized interface insulates the application from the underlying file formats and strengthens the program's portability as well as the data's compatibility.

The MMIO Manager primarily supports the following features:

Buffered I/O: Maintains an I/O buffer that holds data as it is read from (or written to) disk.

RIFF file I/O: Provides simple functions to locate, create, enter, exit, and access the IBM and Microsoft recommended Resource Interchange File Format (RIFF) chunk.

Memory file I/O: A block of memory that looks like a file to an application.

Compound file I/O: A RIFF compound file can contain multiple file elements. The MMIO Manager provides services to find, query, and access any file elements in the compound file.

Installable I/O procedure: A function that knows how to open, read, write, seek, and close files. This allows the application to access non-RIFF files by installing non-RIFF (e.g., AVC™ and TIFF) I/O procedures.

The MMPM/2 primarily supports the RIFF. The following lists the file formats that either the OS/2 base system or the MMPM/2 will support.

- Bundle (BND) File Format
- AVC Audio File Format (AVCA)
- AVC Image File Format (AVCI)
- Musical Instrument Digital Interface (MIDI) File Format
- RIFF MIDI File Format (RMID)
- M-Motion Still Image File Format (MMOT)
- Waveform Audio File Format (WAVE)
- OS/2 1.3 Bitmap File Format (OS13)

Due to the unique characteristics of file services, the MMIO services of MMPM/2 can be used as stand-alone and/or separate from the other MMPM/2 services. The advantage is that applications can now utilize MMIO for pure data object processing.

Brad Noe, *IBM Corporation, Entry Systems Technical Division, P.O. Box 1328, Boca Raton, FL 33429. Mr. Noe is a staff programmer in Multimedia Software Development. He joined IBM in 1983 and has been involved in various software projects including Videotex. He has been in multimedia software development for about a year and a half. Mr. Noe holds a BS in computer engineering from the University of Florida and is pursuing an MS in Computer Science from Florida Atlantic University.*

Bill Lawton, *IBM Corporation, Entry Systems Technical Division, P.O. Box 1328, Boca Raton, FL 33429. Mr. Lawton is a staff programmer in the Multimedia Software Development department in the ESD development laboratory and is the lead designer and developer of the MMPM/2 SPI Subsystem. He received a bachelor's degree in Computer Science from Purdue University. He is currently pursuing a Master's degree in Computer Science from Nova University.*

Multimedia and Graphics

GDDM-OS/2 Link

John Kinchen

by John Kinchen and David Kerr

GDDM (Graphical Data Display Manager) is IBM®'s primary series of mainframe software products that generate and display graphics, image, and text on selected IBM graphics workstations. A workstation running OS/2® Extended Edition Version 1.2 or later together with GDDM-OS/2 Link Version 1.0 can be used as an interactive graphics terminal, giving users access to a large number of mainframe GDDM application programs, and host functions. GDDM-OS/2 Link was developed by a small development team at the Hursley UK Development Laboratories.

HIGHLIGHTS OF GDDM-OS/2 LINK

- It combines the functions of a personal computer and a host graphics terminal in a single workstation. It runs in an SAA™ environment under OS/2 Presentation Manager® windows.
- It is delivered to the workstation through downloading from a host computer, and is serviced in the same way.

David Kerr

Figure 1. Complex Graphics Displayed in Multiple LT Sessions

- It is tightly coupled to the OS/2 Extended Edition 3270 Emulator, providing an integrated user environment. It supports host graphics on all configured 3270 host sessions
- It is capable of supporting any display device for which a Presentation Manager device driver with graphics support is available. It also uses Presentation Manager printer and plotter support to access a wide variety of hardcopy devices.
- It can be used to create Picture Interchange Format (PIF) files and Presentation Manager metafiles on the workstation from host graphics pictures.
- It supports the copying of graphics to the PM clipboard in bitmap format.
- It supports various screen sizes, including, but not limited to 32x80 and 43x80. The number of screen sizes available is dependent on the attached display device.
- It makes use of GPI retained-mode graphics to increase the performance of GDDM-OS/2 Link. This means that any picture redraws are handled locally without asking GDDM to re-transmit the data. On the occasions when this is undesirable (eg., system memory short, complex picture, etc.), this can be deconfigured by deselecting an option on the GDDM-OS/2 Link HOST GRAPHICS OPTIONS dialog.

GETTING STARTED

Prerequisites

GDDM-OS/2 Link Version 1.0 runs on any system unit that OS/2 Extended Edition will run on, subject to storage availability. The storage requirements for GDDM-OS/2 Link are as follows :

- 350 Kb of hard disk storage
- 270 Kb of RAM

It supports all the display attachments supported by the OS/2 Presentation Manager. It also supports all printers/plotters with graphics support, which are supported by OS/2.

As far as software is concerned, it requires:

- GDDM Version 2.3 or GDDM Version 2.2 (with appropriate APAR fixes)
- OS/2 Extended Edition Version 1.2 or later

Installing GDDM-OS/2 Link

GDDM-OS/2 Link is a GDDM family product and as such is distributed on a host tape to mainframe sites. There are no diskettes to install on the workstation.

A small section of GDDM-OS/2 Link is shipped with the OS/2 Extended Edition product. This is simply a stub which enables the installation of the product from the host. Installation is then performed manually by invoking the HGINST command file from either an OS/2 windowed or full-screen session. The command file installs host graphics support by downloading a number of files from the host. After the files have been transferred, the OS/2 Extended Edition 3270 Terminal Emulator must be stopped and restarted to enable host graphics support. Unlike GDDM-PCLK (support for GDDM on DOS workstations), which requires modifications to the GDDM defaults file, there is no extra configuration required to get started.

Removing GDDM-OS/2 Link

It is possible to safely remove the GDDM-OS/2 Link product from your workstation after installation by invoking the HGREMOVE command file from either an OS/2 windowed or full-screen session.

FEATURES

Interactive Host Graphics

Using GDDM-OS/2 Link a GDDM application can make use of all the facilities of a normal graphics terminal plus some unique features.

Using the Mouse: A GDDM application can use the mouse in two ways, as a locator selection device and as a simple choice device.

When GDDM-OS/2 Link takes over control of the mouse it changes the pointer style, to give the user visual feedback, from the usual arrow pointer to one of the following four graphics cursor styles:

- Black and White Target
- Black and White Cross
- XOR Target
- XOR Cross

Selection is determined by the user with the Alternate Cursor (AltCr) key.

Using the Keyboard: GDDM-OS/2 Link can be used without a mouse, as there is a keystroke for each function. The actual keystroke for each action is configurable by the user with the Communications Manager keyboard remap facility. However for the purposes of this article, where actual keys are given, these are for the US Enhanced Keyboard.

Graphics Cursor: The graphics cursor key cycles through the available choices for mouse and keyboard action when a host GDDM application enables the graphics cursor. When the '+Cr' symbol appears in OIA (Operator Information Area), GDDM receives both mouse and keyboard input. The default key assignment is 'alt+F12'.

Cursor Keys: When the graphics cursor is visible and the '+Cr' symbol is visible in the OIA , the cursor keys move the graphics cursor by one screen pixel in each direction. There are also fast cursor keys available which move the graphics cursor by 10 pixels in each direction.

Alternate Cursor: The alternate cursor key (AltCr) cycles through the available graphics cursor styles when the '+Cr' symbol is visible in the OIA. The default key assignment is 'alt+F11'. See 'Using the Mouse' above for a list of the cursor styles supported by GDDM-OS/2 Link.

Graphics Refresh: The graphics refresh key is used whenever the user wants to have the graphics data redrawn in the logical terminal window. The default key assignment is 'ctrl+del'. (This will only work if the 'Retained graphics' option is selected on the HOST GRAPHICS OPTIONS dialog.)

Printing/Plotting GDDM Graphics

As well as being able to output to any host-attached hardcopy device, GDDM-OS/2 Link can print/plot to any suitable hardcopy device attached either locally to the workstation or across a LAN. It uses the services provided by OS/2 Presentation Manager for hard-copy output, can use any of the graphics device drivers currently installed, and can direct output to any graphics device supported by OS/2 Presentation Manager. This means that GDDM-OS/2 Link can use any new devices as they become available and are supported by OS/2 Presentation Manager.

For a host application to access the OS/2 Presentation Manager hardcopy devices a "nickname" must be setup in the GDDM defaults file. One entry must be made for each device to be used. (*See figure A.*)

These lines define two devices which can be accessed by GDDM applications using the nicknames DEFPRINT and PRINTER1, respectively. It is the TONAME field which determines which OS/2 hardcopy device to use. This field contains the printer name, as defined to OS/2, truncated to 8 characters. However there is one reserved name ADMPMOP, which is used to address the current default printer/plotter selected in the Print Manager.

When the user has elected to output to an OS/2-attached hardcopy device, a dialog appears to give a visual indication that the print job is being transferred to the workstation. It also gives the user an opportunity to cancel the job if desired.

By default, the print job is given a system-generated name which indicates the source of the print request (i.e., which LT and position in sequence). However, if the user wishes to name the print job manually in order to follow its progress through a print queue, this can be achieved by selecting an option on the HOST GRAPHICS OPTIONS dialog. This would result in a dialog appearing that requests the user to input a print job name. In most cases this would not be required; however, there may be circumstances when it would be useful to give a meaningful name to a particular print job, if for instance it was one of many.

Saving Host Graphics Files

GDDM offers a facility known as User Control Mode to all applications. Among other features, it provides a method to transfer the currently-displayed host graphics picture to the workstation. A dialog is presented which gives the user an opportunity to specify both a pathname and filename for this file, as well as which format the file should be in (PIF or metafile). Once on the workstation, this graphics file can be used by other applications, for example the Print Picture and Display Picture utilities shipped with OS/2.

Using the Clipboard

GDDM-OS/2 Link provides limited support for the PM clipboard through the use of the 3270 terminal emulator's mark and copy functions. Any area of the host presentation space may be marked and copied into the clipboard in bitmap format. However the following restrictions apply when copying a bitmap representation of graphics to the clipboard :

- The marked area is always an exact multiple of character cells in size.
- Only the visible portion of a marked area is placed into the clipboard as a bitmap.

GDDM-OS/2 Link gives the workstation user access to GDDM graphics functions

```
ADNMNICK FAM=0,NAME=DEFPRINT,TOFAM=1,TONAME=(*,ADMPMOP)
ADNMNICK FAM=0,NAME=PRINTER1,TOFAM=1,TONAME=(*,PRINTER1)
```

Figure A

National Language Support

When GDDM-OS/2 Link is installed, the national-language-specific files to download are determined wherever possible by the language in which the Communications Manager is installed. This ensures that GDDM-OS/2 Link in most cases will appear seamless with the 3270 Emulator. Currently GDDM-OS/2 Link supports: Canadian French, Danish, Dutch, Finnish, French, German, Italian, Japanese, Norwegian, Portuguese, Spanish, and Swedish.

Automatic Service Update

Service fixes for GDDM-OS/2 Link are shipped on a host tape and are loaded on the host system like other GDDM fixes. They are downloaded automatically to the workstation on the next occasion a GDDM application is started, however they will have no effect until the OS/2 Extended Edition 3270 Emulator is stopped and restarted. This means that all workstations connected to a particular host site always have the same level of product code installed, without any user intervention being required.

During this process, a dialog gives a visual indication that the updated modules are being downloaded to the workstation. It also gives the user an opportunity to cancel the operation. This can be particularly useful if the user has some urgent task to perform. In this event GDDM will attempt to download the files again when a GDDM application is started after the next time the Communications Manager 3270 Emulator is restarted.

LIMITATIONS

- GDDM-OS/2 Link does not support CICS™-Psuedo-Conversational or IMS host environments.
- It does not fully exploit the power of the workstation.
- There are some deviations from Common User Access™ 2 (CUA™2).
- It supports the Presentation Manager clipboard bitmap format for copying graphics to other applications, but it does not support the PM metafile format.
- GDDM-OS/2 Link product-related documentation currently is found across both the GDDM library and the OS/2 library, rather than consolidated in a single library.

SUMMARY

In spite of the limitations outlined in the previous section, GDDM-OS/2 Link gives the workstation user access to the wealth of graphics functions available under GDDM. It enables a workstation running OS/2 EE or OS/2 ES to operate as an interactive graphics terminal, and offers GDDM users a migration path from NPT (non-programmable terminals) to OS/2 workstations.

REFERENCES

GDDM Diagnosis and Problem Determination Guide (SC33-0326).

GDDM Guide for Users (SC33-0327).

GDDM Installation and System Management Guide (MVS : GC33-0321, VM : GC33-0322, VSE : GC33-0323).

GDDM Messages (SC33-0325).

IBM OS/2 Extended Edition System Administrator's Guide for Communications (01F0261).

IBM OS/2 Extended Edition User's Guide Volume 2: Communications Manager and LAN Requester (70F0160).

John E. Kinchen, *IBM Hursley UK Development Laboratories, Inc., Mail Point 212, Hursley, UK SO21 2JN. Mr. Kinchen is a project programmer in GDDM Links development. He joined IBM in 1984, and has spent much of his time working on OS/2. He developed the File System application for OS/2 1.1 and has worked on the PM Window Manager and the PM Session Manager for OS/2 1.2, 1.3, and 2.0. He has a Bsc Hon(1st Class) Degree in Computer Science from Portsmouth Polytechnic.*

David A. Kerr, *IBM Hursley UK Development Laboratories, Inc., Hursley, UK SO21 2JN. Mr. Kerr is a project programmer currently on international assignment with IBM's Entry Systems Division in Boca Raton, Florida, working on the architecture and design of Presentation Manager for OS/2 2.0. Since joining IBM in 1985 in Hursley, he has worked on GDDM-PCLK product assurance and GDDM-OS/2 Link design and development. He received a Bsc Hon degree in computer science and electronics from the University of Edinburgh.*

Extended Edition Communications Manager

IBM Extended Services for OS/2 Version 1.0: Communications Manager

Mike Garrison

by Mike Garrison

IBM® Extended Services for OS/2® is a follow-on product to its predecessor, OS/2 Extended Edition (EE) 1.30.1. Extended Services retains enhanced versions of the Communications Manager, Database Manager, and the common Install utility previously packaged with Extended Edition 1.30.1. The LAN Requester has been repackaged with LAN Server 2.0, which now contains both the LAN Requester and the LAN Server. Unlike OS/2 Extended Edition, Extended Services is not packaged with a version of OS/2 Standard Edition (SE). Extended Services will run on separately packaged versions of IBM OS/2 1.30.1 (16-bit), IBM OS/2 SE 2.0 (32-bit) and OEM equivalents.

In addition to the repackaging, functional enhancements have been made to the Communications Manager and Database Manager. Some of the more significant enhancements are highlighted in this article.

OEM HARDWARE PLATFORM SUPPORT

Support for OEM hardware and software platforms extends the scope of Extended Services to run on selected OEM PC hardware, running with OEM versions of OS/2 compatible with IBM OS/2 SE 1.30.1 or higher. A certification process has been put in place to qualify OEM PC hardware and software.

SNA ENHANCEMENTS

Significant enhancements have been made to the System Network Architecture (SNA) support available with Extended Services.

Advanced Peer-to-Peer Networking (APPN)

Advanced Program to Program Communication (APPC) has been enhanced to improve the usability and performance of SNA network operations. These functional enhancements represent the integration of Extended Services APPC capabilities with the Advanced Peer-to-Peer Networking (APPN) capabilities currently available with the IBM Networking Services/2 product. Specifically, these enhancements include:

- Significantly faster APPC throughput than with previous versions of the Communications Manager.
- End node capability, which allows the local node to connect as an end node to a serving network node for directory and routing services.
- Network node capability, which provides network services on behalf of connected end nodes.
- Directory services, which provide the capability to learn the location of partner logical units (LUs) dynamically, rather than having to configure them.
- Route selection services, which calculate the best route for a session through an APPN network.
- Intermediate session routing, which allows two LUs that are non-adjacent nodes to be in session with each other, where the session traverses one or more intermediate nodes.
- Connection network services for the LAN, which provides the capability to learn

LAN destination addresses without having to configure them. (This allows a node to establish a link directly to another node with no LAN destination address configured.)

Also, configuration has been simplified with the following additions:

- IBM supplied modes and classes of service.
- Implicit creation of partner LU and mode definitions when a conversation is allocated.
- Implicit initialization of session limits (the CNOS function), which removes the requirement to explicitly initialize the session limits before conversations can be allocated.
- Removal of session limits for local and partner LUs, which allows greater freedom in configuring and initializing mode session limits.
- Defaults for transaction program (TP) names, operation, and type, which remove the requirement to configure TP definitions for applicable programs.

APPC Usability Improvements

Other enhancements to APPC that make it easier to use include:

- Two new verbs for sending one-way data: SEND_CONVERSATION and MC_SEND_CONVERSATION.
- Local/remote transparency, using intra-node and intra-LU sessions which allow a partner LU to be located in the local node or a remote node.
- Connectivity improvements for LU 6.2 using BIND and session segmenting, which allow the maximum BIND RU size or session data RU size to be larger than the information-field size for the link.
- Upward compatibility for Extended Edition Communications Manager APPC programs at executable file and source file levels.

Common Programming Interface – Communications (CPI-C)

The System Application Architecture (SAA)™ Common Programming Interface for Communications (CPI-C) is available with Extended Services. This provides a portable communications programming interface and supports all CPI-C calls, including additional calls for program-supplied security information.

Management Services (MS) Enhancements

Significant enhancements have been made with respect to Management Services (MS), including application programming interface additions, which allow MS entry point applications to send and receive MS data and receive status information about MS focal points.

Service Point Application (SPA) Router and Remote Operations (ROP) Service

Extended Services contains two functions that provide for network management from the host-based NetView®/370 program. These two components are Service Point Application (SPA) Router and Remote Operations (ROP) Service. A Netview operator can initiate commands from the Netview console that are in turn processed on the OS/2 workstation, with the standard output generated by the command returned to the Netview console on the host. SPA Router and ROP Service help facilitate the following:

- Support of multiple local area networks (LANs) and multiple physical units (PUs)
- Administration of large area networks
- System administration
- Administration of different domains

SPA Router is an OS/2 program that receives the command from the NetView program and sends it to the specified OS/2 application. The application can be any OS/2-based program that runs in protected mode. The advantage of having a separate program, the SPA Router, that directs the commands to the applications, is that multiple OS/2 applications can receive commands from NetView concurrently.

ROP Service is an application that processes (on the OS/2 workstation) the commands sent by the NetView program through the SPA Router. The commands send to ROP Service may be any OS/2 commands that have a command line interface and that do not need interactive user input. In addition to using ROP Service, you can also send commands from the NetView program through the SPA Router to IBM LAN Manager Version 2.0 and IBM LAN Network Manager Version 1.0, and you can use the application programming interface (API) for SPA Router to develop your own applications.

Improved SNA Gateway Support

The LAN DLC now supports up to 255 link stations. As a result, the SNA Gateway support is no longer restricted to 64 concurrently active workstations. The maximum number of concurrently active workstations supported for an SNA Gateway is now 254. However, successful operation in a given environment may depend on other factors such as application load and/or line speed. Therefore, do not assume that you can use 254 workstations for all environments. This change also allows APPC transaction programs to support a larger number of links.

SNA Gateway LU Pooling

The LU pooling algorithm for the SNA Gateway has been improved. The new Extended Services pooling algorithm does not assign pooled LUs to a workstation when it brings up its link to the SNA Gateway. The Gateway assigns pooled LUs only to those sessions that are *started* by the end-user.

For example, the workstation may be configured for five sessions through the SNA Gateway, but the user may have started only two sessions. Gateway will only assign two pooled LUs (instead of five) to the workstation. Individual sessions (instead of all sessions) at the workstation will be automatically logged-off if the session is idle for a configured amount of time. Assigned pooled LUs will not remain assigned until the workstation drops its link to the SNA Gateway. Instead, the pooled LUs are released by the SNA Gateway when the user at the workstation stops a session.

Persistent Verification (PV)

Extended Services delivers the Persistent Verification (PV) enhancement, which supports a sign-on of a user from a local LU and an initial user verification by a remote LU. The results of the user verification persist across subsequent conversation requests from that user to that remote LU until the user is signed off from that remote LU. Extended Services provides PV send support only.

Reactivation After Adapter/Link Failure

When an adapter or link failure occurs, APPC will attempt to restart the adapter every minute for the first 15 minutes and then once every 5 minutes, until the adapter becomes active again. This restart mechanism supports SDLC, Twinax® and LAN connections. X.25™ links are not supported by this mechanism. However, a sample program shipped with the optional Communications Manager Sample Program Diskette provides a measure of automatic link reactivation for X.25 links.

API to Stop Communications Manager

Many of our customers have been needing a programmatic way to stop Communications Manager. This has become increasingly important as we move into more automated forms of systems management. A new API that provides the means for a program to stop the Communications Manager is documented in the *System Management Programming Reference*. A sample program (source and executable) which uses this API is available on the Communications Manager Sample Program Diskette.

LAN ADAPTER AND PROTOCOL SUPPORT

Extended Services contains a re-designed and re-implemented LAN Adapter and Protocol Support (LAPS). The new, performance-oriented NetBIOS™ and IEEE 802.2 ring 0 and ring 3 interfaces retain object-level compatibility with the LAN support provided with earlier versions of Communications Manager. The enhancements provided by LAPS follow.

- **OEM Adapter Enabling.** Extended Services supports the *Network Driver Interface Specification* (NDIS) 1.02 interface. The NDIS interface is supported by many manufacturers of LAN adapters and was developed by Microsoft® and 3COM®. Implementation of the NDIS interface provides OEM adapter support with Media Access Control (MAC) drivers. MAC drivers are device drivers *plugged in* below the NDIS API, that provide low-level access to a communication network adapter. Family 1, Family 2, and busmaster adapters are supported.

- **Increased Adapter Support.** LAPS as shipped with Extended Services supports up to four LAN adapters concurrently. As a result, NetBIOS and IEEE 802.2 applications can concurrently access up to four LAN adapters. However, the Communication Manager features which use the LAN (3270, APPC, etc.) are still limited to two LAN adapters (0 and 1).

- **Presentation Manager® (PM) Interface.** A PM configuration interface is provided to configure LAPS, and is accessed from the pre-existing EZVU-based Advanced Configuration interface.

- **NetBIOS VDD Support.** The NetBIOS Virtual Device Driver (VDD) support allows DOS® NetBIOS applications running in a Virtual DOS Machine (with OS/2 SE 2.0) to share a LAN adapter with other OS/2 and DOS applications.

- **3174 Peer Communications Support.** A MAC driver is supplied with Extended Services which provides 3174 Peer Communications support, also known as LAN over-coax. This function provides peer-to-peer communication for Extended Services workstations connected to an IBM 3174 Control Unit via IBM 3270 communication adapters using coaxial cable.

A bridging function to other LAN segments is also supported. For example, 3174 Peer Communications provides LU6.2, IEEE 802.2, and NetBIOS data flows to other LAN participants via coax attachment. It allows customers to use installed coax to access LAN servers and functions.

EHLLAPI ENHANCEMENTS

Emulator High Level Language API (EHLLAPI) is a feature provided with Extended Services which provides a way for users and programmers to communicate between host and workstations using 3270 terminal emulation and the 5250 Workstation Feature. EHLLAPI provides *programmable keyboard* support for both 3270 and 5250 applications. Functions added to Extended Services include:

- Externalization of the Destination/Origin Structured field services to EHLLAPI.

- Multiple thread support from within a single EHLLAPI application.

- Sharing of 3270 emulation resources between applications.

- Support for invocation of EHLLAPI from stored REXX procedures.

- EHLLAPI support for DOS EHLLAPI programs by providing Virtual Device Driver (VDD) support which emulates Personal Communications 3270.

ACDI ENHANCEMENTS

Asynchronous Communication Device Interface (ACDI) redirection installation has been integrated into the Extended Services install program under Advanced Features. It is also supported via the Custom Install utility.

In Extended Edition 1.30.1, the only method for controlling ACDI redirection was with a command line interface. Extended Services adds an ACDI Redirection API, which allows the redirection of ACDI to be set and queried by interfaces other than a command line interface.

ACDI has also been enhanced in Extended Services to support line speeds up to 64 Kb. This has been accomplished by providing an asynchronous device driver capable of utilizing the high speed DMA data transfer capabilities found in the PS/2 Models 90 and 95.

UTILITIES

A set of utilities are provided with Extended Services. When the utilities are selected for installation, the following programs are installed:

- Programmable configuration (implemented as a REXX extension)
- Trace utility (CMTRACE.EXE)
- SNA trace formatter (FMTTRACE.EXE)
- Display active SNA configuration utility (DISPLAY.EXE)
- Configuration file manager (COPYCFG.EXE)
- Remote display server (RDSPSRVR.EXE)

SUPPORT FOR IBM C SET/2 COMPILER (32-BIT)

By using the IBM C Set/2 Compiler, advanced 32-bit applications can be developed which invoke the following Extended Services Communications Manager APIs:

- APPC (including CPI-C)
- Common service verbs
- System Management verbs
- EHLLAPI
- LAN (NetBIOS and IEEE 802.2)

The C header files provide support for APPC, CPI-C, EHLLAPI, NetBIOS and IEEE 802.2. They can be used with either of the following two compilers:

- IBM C Set/2 Compiler (for 32-bit programs)
- Microsoft C Compiler Version 6.0 (for 16-bit programs)

Programs developed with the IBM C Set/2 Compiler run on OS/2 Version 2.0. Programs developed with the Microsoft C Compiler Version 6.0 run on OS/2 SE Version 1.30.1 or higher (including OS/2 2.0), but will not take advantage of the performance improvements and advanced memory management available for 32-bit programs running on OS/2 2.0.

PRODUCTIVITY AIDS

Included with the Extended Services package are a number of Productivity Aids. These are functions that you may find help make life a little easier in your day-to-day use of the system. However, you should be aware that they are not a formal part of the Extended Services product, and that they are not supported by IBM. These Productivity Aids are provided on an as-is basis without any warranty of any kind.

- ALMCOPY – a flexible, easy-to-use file transfer program.
- TOGGLE – facilitates switching between 3270 sessions.
- PCPRINT – facilitates printing of host and PC files on the PC printer.
- SNAPDUMP – facilitates the collection of problem determination information by capturing the contents of many user-defined files. Includes a PM display facility.
- APL – provides support for displaying and printing APL fonts.
- 3270 Printer Definition Tables – provides a mechanism for creating raw data jobs for printing on the host.

SAMPLE PROGRAMS

The Communications Manager Sample Program Diskette may be purchased separately. It contains these additional sample programs:

- APPC, CPI-C, and System Management Interfaces
- Asynchronous Communication Device Interface (ACDI)
- Server-Requester Programming Interface (SRPI)

- Emulator High Level Language API (EHLLAPI)
- X.25 API (X25)
- Conventional LU Application API (LUA)
- X.25 adapter reactivation sample program
- Stopping Communications Manager sample program
- Sample REXX procedures for programmable configuration
- LAN (NetBIOS and IEEE 802.2)

For 32-bit support, selected SNA, EHLLAPI, NetBIOS, and IEEE 802.2 programs are supplied for use with the IBM C Set/2 compiler. For 16-bit support, Microsoft C Version 6.0 (or equivalent) is supported.

Michael Garrison, *IBM Personal Systems Programming Center, 11400 Burnett Road, Austin TX 78758. Mr. Garrison is an Advisory Programmer for the Extended Services for OS/2 Communications Manager product. He joined IBM in 1979 and initially worked on the 5520 Administrative System and served as the development lead for the SNA Distribution Services (SNA/DS) feature of the 5520. In recent years he has worked as a designer and developer with the IBM OS/2 Extended Edition Communications Manager product. He received a BS in Computer Science from the University of Southwestern Louisiana in 1977, and an MS in Computer Science in 1979, also from the University of Southwestern Louisiana.*

Database Manager

Database Manager: OS/2 Extended Services and DDCS/2

Philip J. Sullivan

by Philip J. Sullivan

This article describes the capabilities and future direction of Database Manager, a component of IBM® Extended Services™ with Database Server for OS/2®, and IBM Extended Services for OS/2. Database Manager is IBM's relational database management system for IBM Operating System/2® (OS/2).

This article also describes the capabilities and future direction of IBM SAA® Distributed Database Connection Services/2 Version 1.0 (DDCS/2)™. DDCS/2 is packaged as two complementary products to the Database Manager component of the Extended Services products.

OVERVIEW

Four new OS/2-based products with relational database capabilities were announced by IBM on October 21, 1991 (*see Figure 1*);

- **IBM Extended Services with Database Server for OS/2** — for multiple workstation database users on a LAN.
- **IBM Extended Services for OS/2** — for a single workstation database user.
- **IBM SAA Distributed Database Connection Services/2 Version 1.0** — for multiple workstation database users on a LAN.
- **IBM SAA Distributed Database Connection Services/2 Version 1.0** — for a single workstation database user.

The Extended Services products consist of a Database Manager component and a Communications Manager component. Database Manager, Communications Manager, LAN Requester, and OS/2 were previously offered as components of IBM OS/2 Extended Edition Version 1.3. As a part of the October 21, 1991 announcements, the LAN Requester was packaged with IBM OS/2 LAN Server Version 2.0. OS/2 will be only offered as separate products: IBM OS/2 Version 2.0 (OS/2 2.0), the 32-bit version, and IBM OS/2 Standard Edition Version 1.30.1 (OS/2 SE 1.30.1), the 16-bit version. Figure 2 shows how the Extended Edition Database Manager and Communication Manager components are packaged with Extended Services.

The Extended Services and DDCS/2 products run in 16-bit mode on OS/2 2.0 and OS/2 SE 1.30.1. They will be available in April 1992.

Database Manager is an IBM Systems Application Architecture® (SAA) relational database management system that supports the SAA Structured Query Language (SQL)™. Database Manager offers a robust application development environment and run-time platform for a variety of OS/2, DOS® and Microsoft® Windows™ 3.0 database applications.

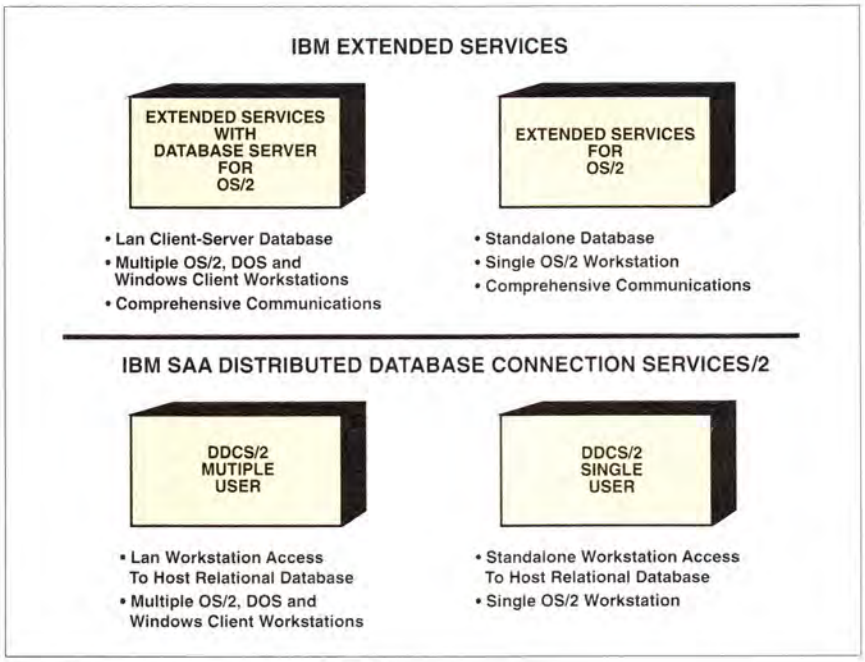


Figure 1. Extended Services and DDCS/2 Program Products

The Database Manager component of the Extended Services products include functions of the Database Manager component of OS/2 Extended Edition Version 1.3, plus these significant enhancements:

- Client-server database and standalone database packaging.
- Support for selected IBM-compatible personal computer systems.
- Access to host relational databases with DDCS/2.
- Forward recovery.

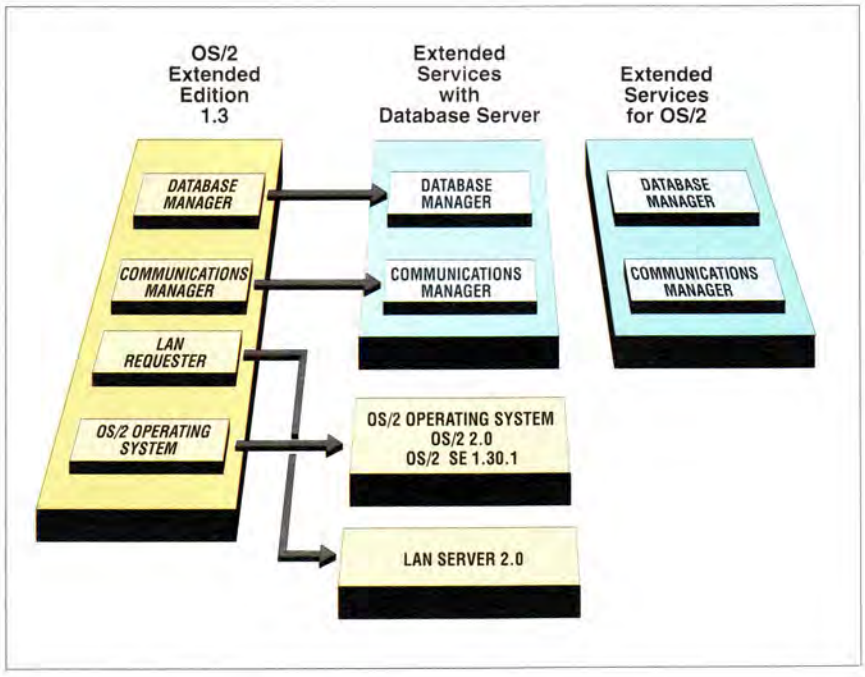


Figure 2. Extended Edition and Extended Services Packaging

- Microsoft Windows 3.0 support.
- NetBIOS® support for OS/2 database clients.
- Database tools enhancements.
- SAA relational database compatibility enhancements.
- Translate character data.

DDCS/2 provides the distributed database connection services to enable OS/2, DOS, and Windows 3.0 database applications to transparently access remote heterogeneous host relational databases that support the IBM Distributed Relational Database Architecture (DRDA)™. Supported databases include IBM DB2® Version 2 Release 3, IBM SQL/DS™ Version 3 Release 3, and IBM OS/400™ (database manager) Version 2 Release 1.1.

Communications Manager includes comprehensive communication functions to connect personal computers with one another and with host computer systems. End-user emulator support and tools for the application developer and the system administrator are also included.

DATABASE MANAGER ENHANCEMENTS

The Database Manager enhancements are a response to customer needs and are focused in the following areas: connectivity and interoperability, data protection, usability, and flexibility.

Client-Server Database and Standalone Database Packaging

IBM Extended Services with Database Server for OS/2 provides database server capabilities for multiple client workstations. A distributed feature, called Database Client Application Enablers, is associated with IBM Extended Services with Database Server for OS/2. A distributed feature may be acquired, for an additional charge per workstation, to be installed on a database client workstation. The distributed feature enables a client to access data in an Extended Services database server on a LAN. The distributed feature includes the OS/2, DOS, and Windows 3.0 Database Client Application Enablers and NetBIOS communications for the LAN. The same distributed feature, used in conjunction with DDCS/2 and Communications Manager, allows a database client read and update access to a remote heterogeneous host relational database that supports DRDA.

IBM Extended Services with Database Server for OS/2 is designed to function as a database server for several users who need to share database information on a LAN, or need to share database information on multiple connected LANs, and/or a need to share information in a remote host database, such as DB2, SQL/DS, or the OS/400 database manager.

IBM Extended Services for OS/2 provides database and communications functions for a standalone workstation. This product is designed for use on an OS/2 workstation by a user who needs a local personal database, and/or a need to access a remote database server, and/or a need for communications functions.

IBM Extended Services for OS/2 also includes the OS/2, DOS, and Windows 3.0 database client enablers and the necessary OS/2 NetBIOS communications to connect to a remote database server on a LAN. An OS/2 database client, with DDCS/2 and Communications Manager installed, may also access a remote heterogeneous host relational database.

Support for Selected IBM-Compatible PCs

Extended Services and DDCS/2 run on selected IBM-compatible PCs. IBM will certify and support Extended Services and DDCS/2 on systems from the following vendors: AST Research®, Compaq®, CompuAdd®, NCR®, Olivetti®, Siemens/Nixdorf®, and Tandy/GRiD®. Extended Services and DDCS/2 may also be supported by other IBM-compatible system vendors. System vendors should be contacted directly to determine if Extended Services and DDCS/2 will be supported.

Support for non-IBM hardware systems extends the versatility of Database Manager. This support provides customers with additional flexibility in selecting hardware platforms: IBM Personal System/2 solutions, non-IBM hardware solutions, or mixed IBM and non-IBM hardware solutions.

Access to Host Relational Databases with DDCS/2

DDCS/2 provides the heterogeneous Remote-Unit-Of-Work (RUOW) functions to allow OS/2, DOS, and Windows 3.0 database clients to read and update IBM SAA host relational databases, as well as have the capability to read and update non-IBM relational databases that support the IBM SQL CPI and DRDA protocol. Note that IBM will not compatibility test, nor make a claim of compatibility with non-IBM relational databases.

DDCS/2 provides the bridge or gateway between Database Manager applications and host relational databases. DDCS/2 converts the Database Manager distributed data protocol to the DRDA protocol supported by the host relational database. The conversion to the DRDA protocol allows OS/2, DOS, and Windows 3.0 database clients to transparently access a single remote host relational database per unit-of-work. DDCS/2 also provides a Database Manager client application with the capability to serially access multiple remote host relational databases.

The multiple-user version of DDCS/2 provides several client workstations running one or more OS/2, DOS, and Windows 3.0 database applications, or multiple concurrent OS/2 database applications running in multiple OS/2 processes, with concurrent connectivity and RUOW access to a DRDA

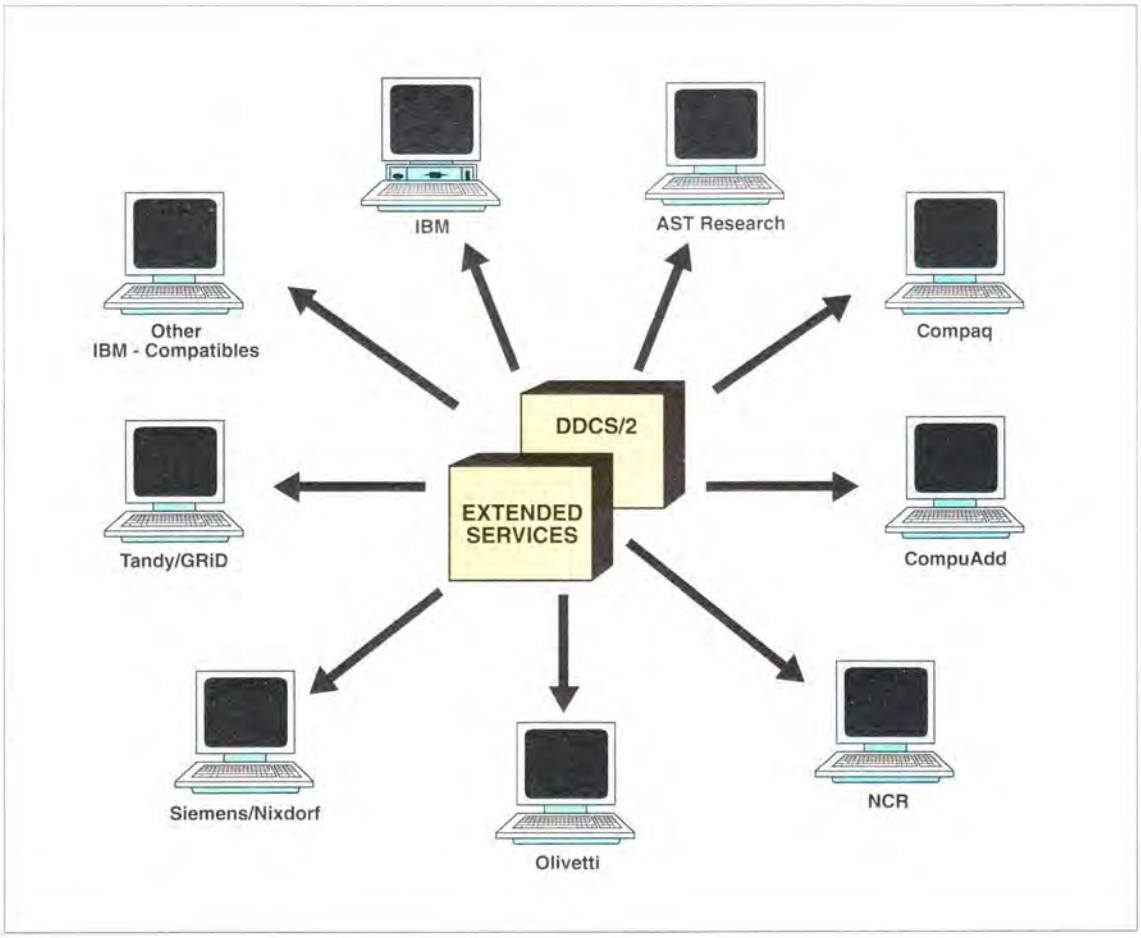


Figure 3. Extended Services and DDCS/2 Support

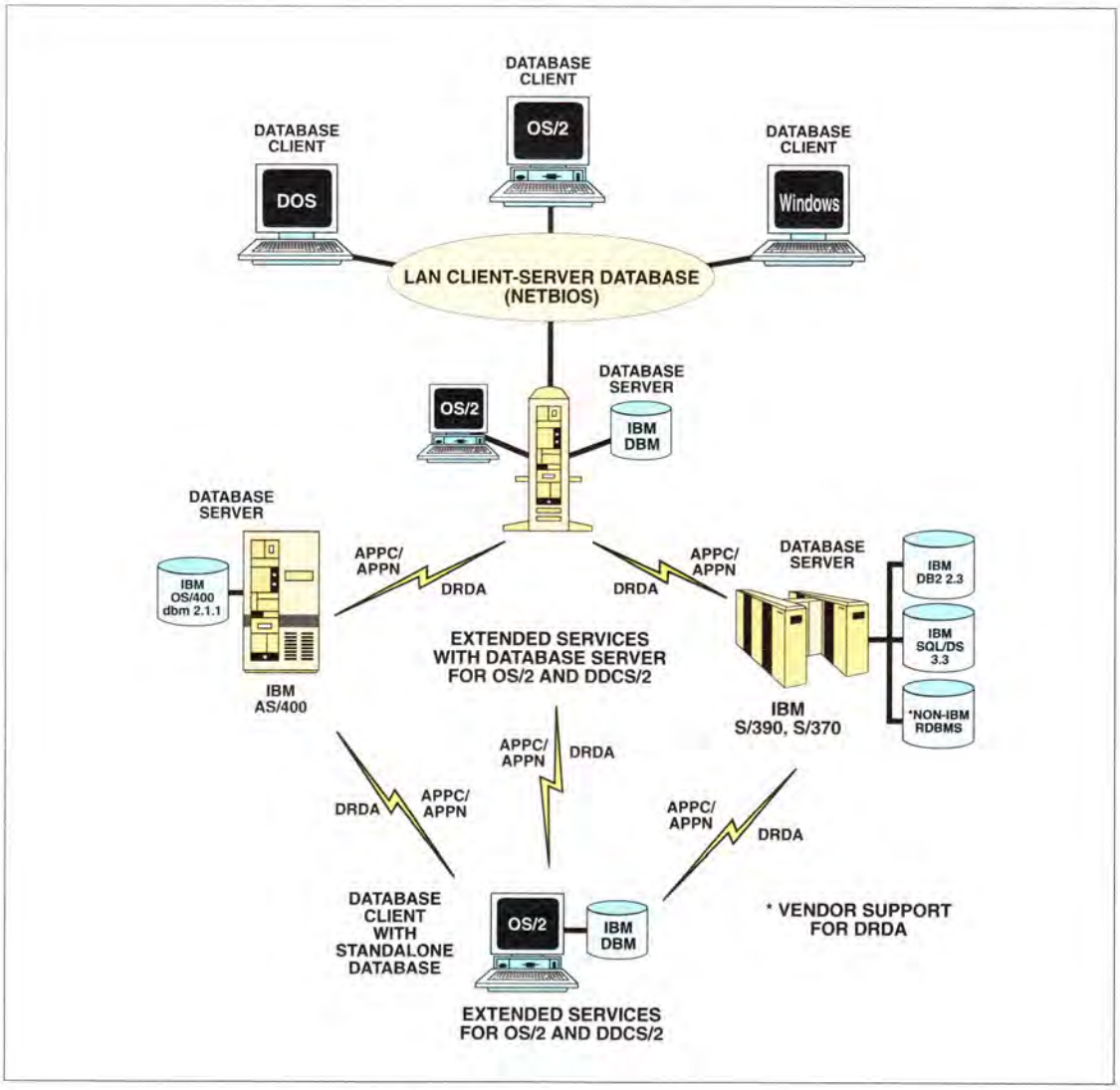


Figure 4. Workstation Access to Host Relational Databases

supported host relational database. The multiple-user version is a cost-effective solution designed to fulfill the needs of several users for access to a remote host database where the client workstations are configured as part of a LAN.

The multiple-user DDCS/2 must be installed on a database server workstation running IBM Extended Services with Database Server for OS/2. The multiple-user version requires Database Manager to be installed on the hardware system.

The single-user version of DDCS/2 provides one client workstation running one OS/2 database application, or multiple concurrent OS/2 database applications running in multiple OS/2 processes, with concurrent connectivity and RUOW access to a DRDA supported host relational database.

The single-user DDCS/2 product is designed to meet the needs of the individual user where the user's workstation is configured in one of the following ways:

- Standalone personal computer workstation that is attached to a host, such as a System/390™, with a need to be connected to a remote host relational database, such as DB2.

- LAN database client workstation that has a need to directly connect via communications to a host relational database due to volume and/or performance requirements.

The single-user version of DDCS/2 may be installed on a standalone database workstation running IBM Extended Services for OS/2, or DDCS/2 may be installed on a database server workstation running IBM Extended Services with Database Server for OS/2. The single-user version requires Database Manager to be installed on the hardware system.

Forward Recovery

Data protection is essential to customers who have mission critical database applications, such as an online order entry application. Database Manager's data integrity and recovery capabilities have been significantly enhanced through the addition of Forward Recovery functions.

Forward Recovery provides the user with the capability to recover lost online data due to a media failure, such as a disk crash. During normal online database transactions, information necessary to maintain data integrity and consistency is preserved in log journals. Journals are normally kept on separate media from the database. Should media failure occur, a backup off-line tape or disk-copy of the database may be loaded on the system and restored as the online database. Forward Recovery provides the means to apply log journal information against the restored database. Log journals contain changes made to the online database since the last backup (off-line copy) of the online database was made. After forward recovery has been completed, the state of the online database is as it was prior to the media failure and subsequent loss of data.

Forward Recovery enables Database Manager to be used for a variety of mission critical applications. The addition of Forward Recovery is consistent with IBM's direction of providing leadership database functions.

Microsoft Windows 3.0 Support

Database Manager Remote Data Services (DOS Database Requester API) functions have been enhanced to provide support for applications developed for the Microsoft Windows 3.0 environment. The addition of this new function provides Windows 3.0 client applications, written to the Database Manager SQL CPI, with transparent RUOW read and update access to a remote Database Manager server attached to a LAN. Windows 3.0 applications will also be capable of reading and updating a remote host relational database, such as DB2, when DDCS/2 is installed on the Database Manager server.

NetBIOS Support for OS/2 Database Clients

NetBIOS Support for OS/2 Database Clients is a new feature of Database Manager. NetBIOS will also be the default LAN communications connectivity protocol for OS/2, DOS, and Windows 3.0 database clients to a remote database server. NetBIOS, as the common LAN communications protocol, will simplify the installation and configuration of database clients and the database server. NetBIOS requires less RAM than APPC/APPN, which reduces the amount of memory required for client workstations.

Database Tools

Database administration tools have been separated from Query Manager and are provided as a separate installable option. Database Tools include functions to configure Database Manager and a database, backup and restore a database, perform forward recovery, and catalog databases and workstations for client-server and DDCS/2 connections. Database Tools have graphical user interfaces that provide an easy-to-use prompted interface which is based on the SAA Common User Access (CUA)™ guidelines and that takes advantage of the OS/2 Presentation Manager® windowing and graphic services.

Command Line Interface

A new command interface is also being provided. Database Command Line Interface

provides the user with the capability to create and run SQL statements, database environment commands, and database utilities from the OS/2 command prompt, and from OS/2 command files. Included with the Database Command Line Interface is a facility (RE-ORG Check) to inform the database administrator when it is beneficial to reorganize a table in the database to improve performance.

SAA Relational Database Compatibility Enhancements

SAA Enhancements improve Database Manager compatibility with the IBM family of SAA relational database products.

SQL Date and Time Arithmetic and Scalar functions provide applications with the capability to add and subtract the following data types: DATE, TIME, and TIMESTAMP. An example of the use and value of SQL DATE, TIME, and TIMESTAMP is an application designed to track and monitor the elapsed time of a commercial airline flight from the originating city and gate to the destination city and gate.

SQLSTATE provides applications with diagnostic information that is common across the family of IBM SAA relational database management systems. This capability makes possible the development of error handling routines which are consistent with and portable across the SAA database family.

User-Defined Collating Sequence

User-Defined Collating Sequence will allow the database user to specify the method which will be used to sort and compare character data. This option may be used to ensure collating results which correspond to IBM host systems, such as System/390.

Translate Character Data

Translate Character Data allows the user to transform character data in various ways; in particular, it makes possible the conversion of characters to their uppercase or lowercase equivalent. This capability makes case insensitive searches possible.

DATABASE MANAGER — HERITAGE AND DESCRIPTION

Database Manager has benefitted from IBM research-developed database technologies and architectures, including the relational model of data and SQL. These IBM technologies and architectures, coupled with the new technologies and architectures developed expressly for Database Manager, have positioned Database Manager to be an industrial-strength relational database providing high performance, concurrent data access, robust data integrity and protection facilities, and consistency with the family of IBM SAA relational database management systems.

Worldwide Product

Database Manager is designed for use as a relational database management system for many world cultures and nationalities.

Versions of Database Manager will be available in thirteen languages. The United States English version will be translated into Canadian-French, Dutch, Finnish, French, German, Italian, Japanese, Norwegian, Portuguese, Spanish, Swedish, and United Kingdom English.

Database Manager supports most of the OS/2 code pages. Code pages provide application developers freedom to match the database system to the configuration of the OS/2 installation. For example, code page 850 may be used to allow for data interchange across all Latin-1 (Western European) languages. Additional cultural support is provided for various date and time formats and collation sequences. With the availability of Extended Services, Database Manager has added support for the Icelandic keyboard and character set (code page 850).

Relational Model of Data

Database Manager is predicated on the relational model of data. The relational model of data was invented by IBM at the San Jose Research Center in the late 1970s. Today, the relational model has been widely accepted as

an industry standard by the marketplace. The relational model has been designed to be easy to understand and use. Information is presented to the user in an easy to recognize and easy to use table format. A table is a logical data structure consisting of rows (records) and columns (fields). The user defines and accesses data in terms of tables and performs operations on these tables.

The main advantage of the relational database model is the clear separation between the user perception of data, i.e. tables consisting of columns and rows, and the internal implementation of data, which is hidden from the user. The simple table format, along with SQL and high-level application development tools, means that the application developer and the database administrator do not have to understand complex physical data structures and access methods, which are characteristics of hierarchical file management systems. The benefits are ease of use and productivity.

Structured Query Language

SQL is the common interface to Database Manager and all IBM SAA relational database management systems. SQL, originally developed at the IBM San Jose Research Center, has also become a standard in the industry. SQL is a high-level data definition and data manipulation language, used for defining, accessing and changing data in tables.

SQL is considered simple to learn, yet powerful in expressing sophisticated queries. A single statement in SQL can perform the same function as many lines of application code developed with a conventional programming language, such as: C, COBOL, or FORTRAN.

All data accesses to Database Manager are performed with SQL statements through the SQL common interface. SQL supports arithmetic operations on retrieved values. The query functions support selective retrieval from single or multiple data tables and dynamic sorting of the set of resulting rows. Built-in functions include summation, grouping, ordering, and basic statistics (for example, calculating an average of the values in a column).

SQL statements can be entered interactively through Query Manager and embedded in a precompiled application program written in a high-level programming language, or embedded in an application program written with IBM Procedures Language 2/REXX. Database Manager provides language precompilers to prepare embedded SQL statements for subsequent application program compilation and execution by Database Manager. Precompiler support is provided for C, COBOL, and FORTRAN.

Database Manager supports both Static SQL and Dynamic SQL statements embedded in an application program. The difference between Static SQL and Dynamic SQL depends on how compilation of the statements occurs at the database. For Static SQL statements, the compilation occurs during precompiling or binding of the application to Database Manager. The compilation is done only once, no matter how many times the statements are executed during the course of running the application. Dynamic SQL statements are compiled each time the statements are executed during the course of running the application. Static SQL is a more efficient process and will, in most instances, provide better performance than Dynamic SQL. However, if the application must issue arbitrary SQL queries, Dynamic SQL is required.

Enhanced Performance

IBM database technologies have been applied to enhance performance. Record Blocking is a technique that returns blocks of records instead of one record at a time to the requesting application. The Query Optimizer automates the selection of efficient access paths to data. The access paths include the use of sorts and join techniques and indexes. Row Level Locking allows a high-level of concurrent access to data and is especially important to database transaction processing performance. Granular levels of data isolation are provided to enhance performance. These include Uncommitted Read, Cursor Stability, and Repeatable Read.

The Database Manager Application Remote Interface (ARI) may be used to enhance performance. It allows a developer to write an application program where the application processing can be split between the database client and the database server on a local area network. When the application is run, some of the application processing load can be transferred from the client to the server, resulting in a reduction of data traffic on the LAN and a significant improvement in database application performance.

Robust Data Integrity and Protection

Database Manager has extensive data integrity and protection features to preserve the integrity of data, thereby ensuring users that information being accessed is consistent.

Transaction Management

Transaction Management is a feature that allows multiple applications to run concurrently against common tables in Database Manager. Through the use of transaction management, Database Manager provides a high level of data control and protection against serialization problems and database transaction failures in a multitasking, multiuser database environment. If an application accesses Database Manager and terminates normally, a COMMIT statement is issued which allows the database updates to become permanent. If an application is interrupted in the midst of a transaction, the system can perform a ROLLBACK on all uncompleted work after an application failure, or on the next system restart after a system failure. These functions help ensure the integrity and consistency of the database information.

Database Manager provides transaction management through the use of locks, multiple levels of data isolation, and a Recovery Log. The lock function and the specified level of data isolation are used to prevent another application from updating a data record while a transaction is pending against that record. All changes to data tables and indexes have entries written to the Recovery Log that provide sufficient information to allow Database Manager to back out of an update before any changes are written to the data. Through the use of the Recovery Log, updated during normal database processing using a write-ahead logging scheme, Database Manager can return the database to a consistent state after system failures, such as a power failure, a user-initiated system reset, or a software failure (except in some instances where logical media failure has occurred). In the event of such a failure, Database Manager can undo and redo the necessary database operations to return the database to a consistent state by retrieving and reconstructing committed data and rolling-back uncommitted data using the Recovery Log.

Restore and Recovery Capabilities

Database Manager is capable of restoring a backup copy of a database, where data on the media, disk or diskette, is destroyed by physical or logical damage. Physical media failure can occur due to a bad sector on the disk or diskette. Logical media failure can occur when data is over-written by another program and becomes unrecognizable to Database Manager. Logical media failure can also occur if a system fails during a data update, such that some sectors are left unchanged while some are updated. Database Manager attempts to recover from logical media failures using the Recovery Log. If this attempt fails, the backup copy of the database must be restored.

After the backup copy of the database has been restored, Forward Recovery provides the means to apply log journal information against the restored database. Log journals contain changes made to the online database since the last backup (off-line copy) of the online database was made. After forward recovery has been completed, the state of the online database is as it was prior to the media failure and subsequent loss of data.

Referential Integrity

Referential Integrity is another important feature used to provide robust data integrity. It ensures the consistency of data values between related columns of different tables.

For example, a user may define an EMPLOYEE table that contains employee and department numbers and a DEPARTMENT table that contains department numbers. In addition, the user may want to ensure that for every department number in the EMPLOYEE table there must be an equal and unique department number in the DEPARTMENT table. Such a constraint defined on the EMPLOYEE table is called a referential constraint. The department number in the DEPARTMENT table is called a primary key, and the department number in the EMPLOYEE table is called the foreign key in this constraint. Enforcement of this constraint provides referential integrity. The Database Manager records and enforces this data relationship, and enforcement by application logic is not necessary.

In addition to ensuring the consistency of data, Referential Integrity can also improve application development productivity by allowing this function to be moved out of each application program and into the Database Manager.

Data Security

In order to access and use objects in the Database Manager, the user must be identified to User Profile Management and be validated by a password on the first use of the Database Manager. The user is then associated with a valid USERID. Access to a specific database and the objects within it (for example, tables, views, access plans) is controlled by SQL GRANT/REVOKE statements. A creator, or other specifically authorized user of a database object (such as a systems administrator or database administrator) may protect the object by only granting access rights to specific users and/or groups. Another user must be specifically authorized to access and update a database object. These rights can also be revoked as required. A creator also has the option to allow public access to all database objects.

IBM SAA Relational Database Consistency

Database Manager is a member of the family of IBM SAA relational database management systems: DB2 and SQL/DS, which run on IBM System/390™ and IBM System/370™ mainframe and mid-range systems; and the OS/400 database manager, which is part of IBM Application System/400™ systems.

Syntax and semantics of the Database Manager SQL interface functions are designed for consistency with the family of IBM SAA relational databases. For example, the consistent handling of host variables, precompile, bind, and query optimization simplifies the application developer's work in writing heterogeneous cross-system applications for an enterprise-wide distributed database environment.

DATABASE MANAGER COMPONENTS

The major components of the Database Manager are Database Services, Remote Data Services, Query Manager, and Database Tools.

Database Services

Accesses to Database Manager are performed by Database Services through the SQL Common Programming Interface (CPI), the Database Manager Command Line Interface and utility application programming interfaces (APIs). Database Services manages the data stored in the database, generates access plans to the data, and includes transaction-management, data integrity and protection, multiple-user concurrency, and security functions.

Database Services also provides a number of utility functions that assist the user in maintaining and manipulating the contents of a database. These utilities include functions to import and export data from an OS/2 or DOS file, save and restore individual data tables, perform database backup and restore functions, and optimize system performance.

Remote Data Services

Remote Data Services provides RUOW functions and database connection services to allow Database Manager to function as a client-server distributed database management

system. These functions are required for database client workstations to access a remote Extended Services database server or a remote host relational database server.

Database clients and database servers may be LAN-attached or non-LAN-attached. Thus, Remote Data Services supports a variety of LAN and non-LAN environments. Supported LAN environments are IBM Token-Ring™, IBM PC Network™, and IBM ETHERAND™. Supported LAN communication protocols are NetBIOS, APPC/APPN, and SQL LAN-Only Option (SQLLOO). APPC/APPN is also supported for non-LAN environments.

Extended Services database servers (IBM Extended Services with Database Server for OS/2) support attachment of the following database client workstations:

- Extended Services OS/2, DOS, and Windows 3.0 clients use the NetBIOS protocol. As an option, OS/2 clients may use APPC/APPN.
- Extended Edition 1.3 and 1.2 OS/2 clients use either APPC or SQLLOO.
- Extended Edition 1.3 and DOS 1.2 clients use NetBIOS.

Remote Data Services, in conjunction with DDCS/2 and Communications Manager, also provides RUOW functions to allow database clients to access a remote host relational database, such as DB2, that supports the DRDA protocol. DDCS/2 provides host relational database access for all clients supported by Remote Data Services. Communications Manager provides the connection from the DDCS/2 (client) workstation to the host relational database using APPC/APPN.

Query Manager

Query Manager provides tools and functions that allow users to access data and manipulate data in Database Manager.

Query Manager provides an easy-to-use, prompted user interface which is based on the SAA Common User Access (CUA) guidelines and takes advantage of the OS/2 Presentation Manager windowing and graphic services.

The prompted interface hides SQL from the application user, so that the user does not have to learn SQL syntax. However, once a prompted query has been created, its SQL syntax can be optionally viewed. This allows users to learn SQL syntax and enter SQL statements in a non-prompted mode.

Query and Report Writing Tools provide functions to create a database query, initiate the data access request, and display the results of the query via the report feature. The user may save the query and report objects and save or print the accessed information. A Business Graphics Interface provides the user with the capability to optionally install and use a third party business graphics program that has been written to this interface. This permits graphic presentation (pie, line, bar, tower graphs, etc.) of report data that was accessed by the Query and Report features.

Custom Application Tools provide functions that allow a user to create custom database applications. Panels may be used to create custom data entry and edit formats. Procedures may be used to create Query Manager commands or procedure language statements to execute previously defined query and form (report) objects, or to combine and automate several panel operations. Menus may be used to create a customized list of application selections. Items listed on the Menu may be selected to initiate and automatically run queries, generate reports, run and call procedures, panels, and other menus.

A Command Line Interface is also provided for users to issue Query Manager commands directly. These commands include functions such as print, get, import, and display. These commands may be used to execute Query Manager objects, such as Procedures, directly from the command line.

Database Tools

Database Tools provide the following functions to administer and manage a database system: creating, altering and dropping tables, views, indexes, entering and editing data, defining referential constraints, importing and exporting data, reorganizing tables, and monitoring database activities.

Operational Status provides a snapshot of information about current database activity. This feature provides information about where the databases are located, alias names, the time and date a database was last backed up, and how many applications are currently connected to a specific database.

FUTURE DIRECTION

Future versions and releases of Database Manager and DDCS/2 will focus on providing industrial-strength distributed relational database solutions for LAN environments, and in concert with other IBM SAA relational database products enterprise-wide distributed relational database solutions for interconnected LANs and host systems. Continued emphasis will be placed on satisfying the needs of the marketplace, including higher levels of data and user concurrency, distributed database connectivity and interoperability enhancements, increased performance, capacity and availability, and database administration, usability, and security enhancements.

Distributed Database Enhancements

Database Manager and DDCS/2 will be enhanced to provide additional distributed database functions. Support will be provided for increasingly more complex data access such as distributed-unit-of-work, distributed (SQL) statements, and distributed tables.

Common 32-Bit Database Manager for OS/2 and AIX

IBM's direction is to provide a common 32-bit client-server SAA/AIX® Database Manager that fully exploits the 32-bit OS/2 and AIX platforms. The common Database Manager will be provided for Intel-based IBM Personal System/2® computers, Intel®-based IBM-compatible personal computers and for IBM RISC System/6000® platforms.

Database Manager for OS/2 and AIX will be optimized for LANs. Emphasis will be placed on portability, interoperability, and conformance to industry and defacto standards.

Parallel Database Processing

IBM's direction is to enhance Database Manager to provide significantly higher levels of performance, capacity and availability. Parallelism will support multiple system processor nodes, such as multiple IBM Personal System/2 computers, or multiple IBM RISC System/6000 computers, that are linked together to present a single database system image.

Database Manager parallel processing functions provide customers with high throughput rates, fast response times, access to very large (gigabytes) amounts of information, and fault tolerant functions. Database parallelism will be especially beneficial to online transaction processing applications where fast response time and high availability are essential. Parallelism will also offer significant benefits to advanced complex-data applications, such as image and multimedia, where access to large amounts of data are application characteristics.

Promote and Support the Development of Database Manager Tools and Applications

IBM will continue to provide Database Manager technical support to complementary independent application developers and to business partners. This support is provided through developer assistance programs. The purpose of these programs is to provide assistance in the design and development of a variety of OS/2 Presentation Manager, Windows 3.0, and DOS database applications and tools, such as general purpose query and report writers and customizing tools, that work with Database Manager.

Make Database Code Available to Selected Software Developers

IBM's direction is to make components of Extended Services and DDCS/2 database code available to complementary software developers who meet IBM's selection criteria. Selected software developers may integrate, package and market database code with

application programs they develop. General purpose database query and report writing programs, and industry specific custom application programs are examples of the types of programs that may use the database code.

Database code can provide the software developer with industrial-strength relational database management services, distributed database connection services to IBM OS/2 relational database servers attached to a local area network, and distributed database connection services to IBM SAA host relational databases.

Philip Sullivan, *IBM Personal Systems Programming Center, 11400 Burnett Road, Austin TX 78758, is a Senior Product Planner. Mr. Sullivan joined IBM in 1966 as a sales representative and has held various professional and management positions in marketing and product planning. Mr. Sullivan has been involved in the product planning of software products for fourteen years. The past three years he has been responsible for the development of IBM Personal Systems relational database market and product strategies for personal computers and advanced workstations. He holds a Bachelor of Science degree in Economics from Mount Saint Mary's College. He also completed a course of study in Marketing Analysis and Planning at the Wharton School at the University of Pennsylvania.*

Database Manager

Using System Catalogs To Optimize Database Applications

by Karen Tyger

The table layout and index structure of a database can have a significant impact on database application performance. This article explains how IBM® OS/2® Extended Edition Database Manager (DBM) stores user, table, and index data, and how to utilize system catalogs to create efficient databases, tune database performance, and determine when database reorganization is needed.

The IBM OS/2 EE (DBM) contains a wealth of statistical information about tables, views, indexes, authorizations, and plans. This information is stored in special database tables called "system catalogs" which are created automatically when the database is created through CREATE DATABASE. An application can query the system catalogs and use the information to analyze database performance. The information in the catalogs cannot be altered via SQL commands; however, performing runstats on a table will update some of the statistics. Figure 1 shows the SQL operations that can be performed on user tables and system tables.

Karen Tyger

DBM CONCEPTS

To understand more about how system tables describe the database objects, you must first understand how the DBM stores data. Each table is stored on the DASD in a separate binary file. The file is divided into 4096 byte pages. As rows are deleted from the tables, some of the pages might become partially empty or perhaps completely empty. However, the DBM reuses as much space as possible when new rows are added to the table.

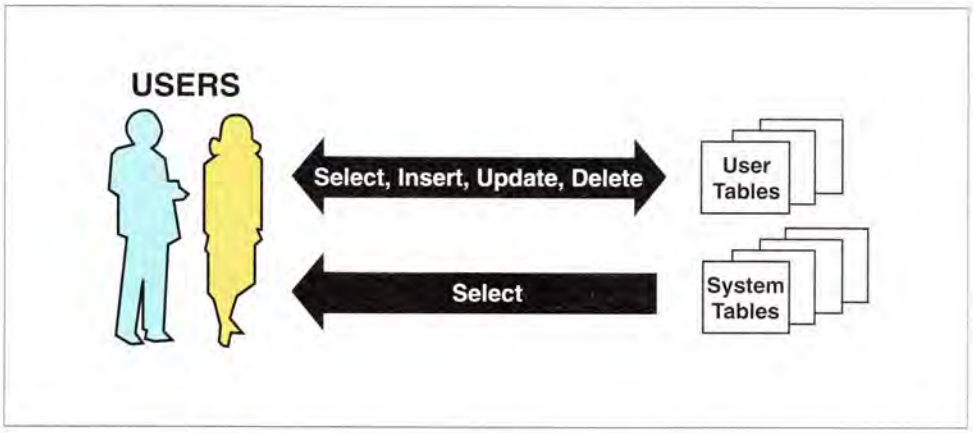


Figure 1. SQL Operations Available Using User Tables and System Tables

When a row is added to a table, DBM stores the data sequentially. The data is packed tightly to minimize unused space. If the data is stored in a fixed length field (CHAR), enough space is allocated for the entire length of the column regardless of how much actual data is stored. Figure 2a shows data stored in fixed length fields. Column 2 is defined as 7 characters long CHAR(7) and contains 5 characters of data. DBM allocated 7 bytes of storage even though the data is only 5 bytes long. The remaining 2 bytes are filled with an ASCII blank. If we want to update the row and store a 7 byte value in this CHAR field, the DBM has already allocated 7 bytes and simply writes over the existing data. Now, the end of the field is not padded with blanks.

DBM handles a varying-length field (VARCHAR) differently (*see Figure 2b*). When inserting 5 characters into a 7 byte VARCHAR field, DBM will allocate 5 bytes and store the 5 characters in this space. Updating the row and storing 7 bytes in this VARCHAR field will cause a problem because there is only enough space for 5 characters. In this case, DBM creates a new record and stores the 7 byte value in this new location. This new record is called an overflow record. Data describing the location of the overflow record is stored in the old record. Another way that overflow occurs is when columns are added to tables. The same principle as explained above would apply in this case.

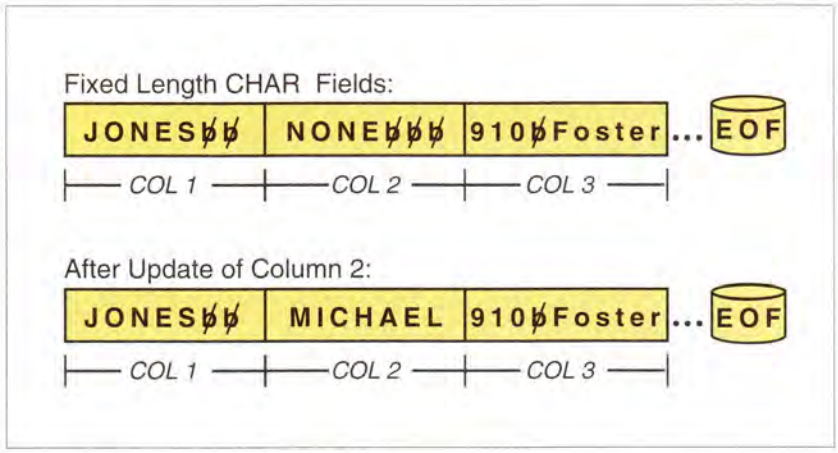


Figure 2a. Data Storage of Fixed-Length Fields

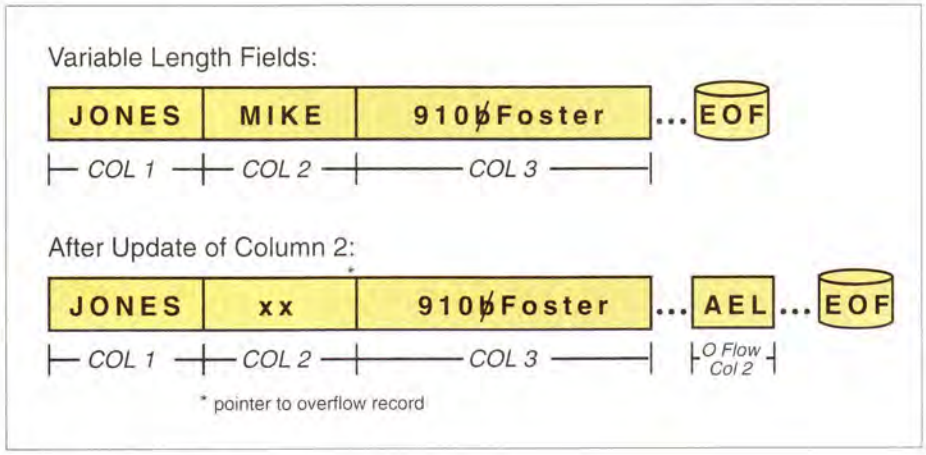


Figure 2b. Data Storage of Variable-Length Fields

The DBM indexes are stored in a binary file also. One index, or "key" file is created for each table in the database. The indexes are organized in a binary tree. Even if a table contains duplicate rows, the key data is stored in the index file only once. However, an entry is made in the index file to track the number of table rows that match each key.

The DBM provides control at a system level, a database level, or database object levels. The database objects can be divided into three areas:

- tables and views
- plan
- index

Control at the system level is obtained by creating a userid that has SYSADM authority. Control at the database and database object level is done using SQL GRANT and SQL REVOKE. Authorizations also can be created during SQL-binding.

SYSTEM TABLES

The system catalogs can be grouped into three areas:

- table statistics
- authorization statistics
- plan statistics

Figure 3 lists the three different groups and the tables that belong in each group. There are seven tables that contain information about table statistics. The SYSTABLES table describes each table or view in the database. The referential relationships between tables are defined in the SYSRELS table. Data about each column defined in each table or view is stored in the SYSCOLUMNS table. A description of indexes is stored in SYSINDEXES. SYSVIEWS and SYSVIEWDEP both contain information pertaining to views. Backup information is contained in SYSBACKUP.

TABLE STATISTICS

SYSCOLUMNS	SYSINDEXES
SYSTABLES	SYSVIEWS
SYSVIEWDEP	SYSRELS
SYSBACKUP	

AUTHORIZATION STATISTICS

SYSBAUTH	SYSPLANAUTH
SYSTABAUTH	SYSINDEXAUTH

PLAN STATISTICS

SYSPLAN	SYSPLANDEP
SYSSECTION	SYSSTMT

Figure 3. System Catalogs Maintained by OS/2 EE DBM

The catalogs contain information about authorization at the database level and the database object level. As explained above, database objects can be divided into 3 areas. There are four levels of authority to access the database and database objects: database, plan, table/view, and index. Therefore, authorization data is located in four system catalogs. Each catalog corresponds to one of the authorization levels. SYSDBAUTH lists database authorization for each user. Plan authorization data is stored in the SYSPLANAUTH table. Table/view authorization and index authorization are stored in the SYSTABAUTH and SYSINDEXAUTH respectively.

Statistics about database plans also are stored in four catalog tables. General information about the plan is stored in SYSPLAN. The dependencies an access plan has on indexes, tables, and views are recorded in SYSPLANDEP. The sections of the access plan are contained in SYSSECTION, and the SQL statements (or portion of) are stored in SYSSTMT.

Appendix D in the DBM SQL Reference (S01F-0293) contains a description of each system catalog.

TABLE INFORMATION

SysTables

The SYSTABLES, SYSCOLUMNS, and SYSINDEXES tables in the table statistics group contain the information needed to analyze database performance. One row is inserted in the SYSTABLES table for each new table or view that is created. The NAME, CARD, NPAGES, FPAGES, and OVERFLOW columns are used in determining if a table needs to be reorganized.

The NAME column contains the name of the table. The total number of rows in the table is stored in a column called CARD. The FPAGES column describes the total number of pages in the table while NPAGES describe the number of pages that contain valid data. As described above, it is possible for some pages to be empty. The OVERFLOW column is the total number of overflow records in the table.

SysColumns

NAME, TBNAME,COLCARD, HIGH2KEYCARD, LOW2KEYCARD, and AVGCOLLEN are some of the fields in the SYSCOLUMNS table. The table name is stored in the TBNAME column. The COLCARD is the number of "distinct" values in the column. The HIGH2KEY and the LOW2KEY are the second highest value in column and second lowest value in column respectively. The highest and lowest values are not used because they may be a flag or temporary value like '99999' or 'zzzz', and so would not correctly represent the range of data stored in the table. AVGCOLLEN stands for average column length. When the data contained in the column is of varying length (VARCHAR), this field can be used to approximate the length of the column.

SysIndexes

The SYSINDEXES table contains NLEAF, NLEVELS, FIRSTKEYCARD, FULLKEYCARD, and CLUSTERRATIO. Remember that database indexes are organized in a binary tree. NLEAF is the number of leaf pages at the bottom of the tree and NLEVELS is the number of levels in the index tree. An index can contain one column or several columns. A full key value describes all the columns that make up the index while a first key value refers to just the first column that makes up a multi-column index. DBM stores the number of distinct full key values in FULLKEYCARD and the number of distinct first key values in FIRSTKEYCARD. The word "distinct" is used to emphasize that only the number of unique values is counted. The quickest measure of an indexes worth is the value of CLUSTERRATIO. The cluster ratio is a percentage from 0 to 100% that measures how closely the index ordering matches the data stored in the table. A high cluster ratio indicates that the index and the data in the table are almost in the same order. A low number indicates that the index and the data are not in the same order. For example, if the data is ordered in descending order and the index is ordered ascending, the CLUSTERRATIO will be 0%.

Authorization Information

The tables that are grouped under Authorization in Figure 3 all have a common table structure. There are columns that describe the userids, the name and creator of the database object, and the privileges. The only exception is that the SYSDBAUTH table does not contain the name/creator of the database object.

The columns containing userid information are named GRANTOR and GRANTEE. The GRANTOR is the ID of the person who granted the privileges and the GRANTEE is the ID of the user who holds the privileges. The database object is described in two columns which are NAME and CREATOR for SYSPLANAUTH and SYSINDEXAUTH, and TTNAME and TCREATOR for SYSTABAUTH. The NAME is the name of the database object and the CREATOR is the qualifier of the database object. A column is created in each table for each privilege. The value of the column is either "Y" (user holds the privilege) or "N" (user does not hold the privilege).

One or more rows are inserted for each user that is granted a privilege. Why not just have one row per userid? Since GRANTOR information is contained in the row information, each unique grantor will cause a new row to be inserted into the database. For example, when userid DBAKAREN successfully grants userid BETH CONNECT authority, a row will be inserted into the SYSDBAUTH table. Now, if DBADEE grants BETH CREATETAB authority, another row will be inserted into the SYSDBAUTH table because the GRANTOR information is different. There will be a total of two rows in the database describing the database level authorizations for BETH. If anyone needs to know who granted BETH that extra CREATETAB authority, the information is stored in the system table!

The SYSDBAUTH table holds database level authorizations. It contains six columns: GRANTOR, GRANTEE, DBADMAUTH, CREATETABAUTH, BINDAUTH, and CONNECTAUTH. The DBMADMAUTH column documents whether the user has DBM administrative privileges or not. A user can create a table if the value of CREATTABAUTH is "Y". The privilege to bind is described inthe BINDAUTH column, and a user can open the database if the CONNECTAUTH field is set to "Y". Similarly, the SYSPLANAUTH contains information about the user's privileges for a single plan.

The SYSPLANAUTH table holds access plan level authorizations. It contains seven columns: GRANTOR, GRANTEE, NAME, CREATOR, CONTROLAUTH, BINDAUTH, and EXECUTAUTH. If the user holds control privilege over the plan, CONTROLAUTH will contain a "Y" value. Users can drop, bind, or execute a plan if they hold CONTROL privilege. BINDAUTH is the authority solely to bind an access plan to the database and EXECUTEAUTH is the authority solely to execute the access plan.

Table and view level authorities are stored in a twelve-column table called SYSTABAUTH. As before, the table contains the GRANTOR, GRANTEE, TCREATOR, and TTNAME columns. The names of the columns that determine user privileges are CONTROLAUTH, ALTERAUTH, DELETEAUTH, INDEXAUTH, INSERTAUTH, SELECTAUTH, UPDATEAUTH, REFAUTH. These columns determine whether the user can assert these privileges on a table or view: control, alter, delete, insert, select, update or reference (in a referential constraint).

The SYSINDEXAUTH table holds index level authorizations. It contains the standard GRANTOR, GRANTEE, NAME, and CREATOR columns. The only privilege column is CONTROLAUTH, which determines whether the user has control authority.

Plan Information

Plan information is stored in four tables; however, this discussion is limited to SYSPLAN. Every time a plan is bound to the database, a row is inserted into the table SYSPLAN. The columns NAME, CREATOR, and BOUNDBY describe the plan name. CREATOR is the userid that performed SQLPREP and created the bind file. The BOUNDBY field contains the userid of the person who bound the plan to the database either by SQLPREP, SQLBIND, or an API call. Two other important columns are ISOLATION and BLOCK. The isolation level of the plan is stored in the ISOLATION column. The values of this field are "R" for repeatable read, "C" for cursor stability, and "U" for uncommitted read. There are three record blocking options which are no blocking "N", block all "B", and block unambiguous "U". The record blocking option is contained in the BLOCK column.

APPLICATION PROGRAMS

Table Reorg Check

An application program that checks table statistics can use the information contained in the system catalogs to determine if a table should be reorged. First, execute RUNSTATs to update the information in the system tables.

Three important table statistics are:

- Page usage
- Space usage
- Overflow

Figures 4a and 4b contain the formulas required to calculate these values.

TABLE FORMULAS

$$\text{PAGE USAGE} = \frac{\text{NPAGES}}{\text{FPAGES}} > .8$$

$$\text{SPACE USAGE} = \frac{\text{Table size}}{\text{Allocated space}} = \frac{(\text{AVGCOLLEN}_1 + \text{AVGCOLLEN}_2{}^{①} + \ldots + 8) * \text{CARD}}{(\text{FPAGES} - 1) * 4020} > .7$$

$$\text{OVERFLOW} = \frac{\text{Overflow}}{\text{Card}} < .05$$

① Add all columns in the table

Formulas derived by Tim Malkemus, IBM Austin

Figure 4a. Table Formulas

INDEX FORMULAS

$$\text{CLUSTER RATIO} = \text{CLUSTERRATIO} > .8$$

$$\text{SPACE USAGE} = \frac{\text{Index size}}{\text{Allocated space}} = \frac{\text{FULLKEYCARD} * (\text{KEY SIZE} + 10) + (\text{CARD} - \text{FULLKEYCARD}) * 4}{\text{NLEAF} * 4096} > .5$$

$$\text{Where Key size} = (\text{AVGCOLLEN}_1 + \text{AVGCOLLEN}_2 + \ldots)^{①}$$

$$\text{LEVELS USAGE} = \frac{\text{90\% Max \# entries}}{\text{Actual \# entries}} = \frac{.9 * \left(\frac{4000}{(\text{Key size} + 10)}\right)^{**(\text{NLEVELS}-2)}}{\frac{\text{Indez size}}{3076}^{②}} < 1$$

① Add all columns participating in index
② Divide by 3076 to approximate # of entries

Formulas derived by Tim Malkemus, IBM Austin

Figure 4b. Index Formulas

Page usage is defined as the ratio of NPAGES to FPAGES. Both numbers are maintained in the database system catalogs (SYSTABLES). This ratio compares the number of pages that contain data to the number of pages that do not contain data. A high ratio indicates that many of the pages allocated for your table are in use. If the ratio is low, there are many empty pages in your table and it should be reorganized. Ideally, at least 80% of the total number of pages allocated should contain data, so this ratio should be greater than .8. Pages can become empty after rows are deleted, as explained above. Notice that this ratio describes the number of pages in use and not the portion of each page in use. To learn that information, you would calculate the space usage ratio.

The space usage ratio describes the percentage of disk space utilized for table data. It is defined as the ratio of table size to allocated space. The table size is calculated for each table by taking the average column length (AVGCOLLEN) found in SYSCOLUMNS, adding 8 bytes, and then multiplying by the total number of rows in the table (CARD). The denominator, allocated space, is determined by multiplying the page size less 76 bytes of header data (4096-76=4020) times (FPAGES -1). (One is subtracted from FPAGES because the last page allocated usually is not filled.) If the ratio is greater than .7, most of the space allocated for the table is not in use and the table should be reorganized.

The overflow ratio measures the number of overflow records in the table versus number of rows in the table. (OVERFLOW/CARD). The number of overflow rows should be less than 5% of the total number of rows in the table. The access time for a row can increase when overflow occurs, because collecting the data might require 2 reads instead of 1. The additional read is for an overflow record stored in a distant place in the table file.

These three ratios are calculated in the C program shown in Figure 5. First the SQL data is collected. Before the ratios are calculated, the program ensures that RUNSTATS has been executed. Next the ratios are calculated. Finally, the ratios are displayed on the screen along with a message declaring if the value of each ratio fell within the accepted range.

```
// Process Tables Statistics

// Retrieve NPAGES,FPAGES,OVERFLOW, and CARD info from SYSTABLES.
EXEC SQL
  SELECT NPAGES,FPAGES,OVERFLOW,CARD into :npages,:fpages,
         :overflow,:card
     FROM SYSIBM.SYSTABLES
     WHERE NAME = 'YOUR_TABLE_NAME';

// Retrieve AVGCOLLEN info from SYSCOLUMNS for each column in table.
// Add all AVGCOLLEN together to obtain the total length for one row.

// DECLARE CURSOR
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT AVGCOLLEN into :avgcollen
     FROM SYSIBM.SYSCOLUMNS
     WHERE TBNAME = 'YOUR_TABLE_NAME';

// OPEN CURSOR
EXEC SQL OPEN C1;
```

Figure 5. C Program to Process Table Statistics (Continued)

```
// While no SQL error, retrieve all data and keep running total of length

while (SQLCODE==0) {
 EXEC SQL FETCH C1 INTO :avgcollen;
 totrowlen := avgcollen + totrowlen;
}

// Page check. Calculate the ratio NPAGES/FPAGES.  80% of pages in
// the table should contain data; therefore, this ratio must be
// greater than ".8".

// SQL returns -1 if RUNSTATS not executed. If RUNSTATS has been
// executed, calculate ratio else print failed msg.

if (npages != -1) && (fpages != -1)
    page_ratio = npages/fpages;                // calculate page_ratio.
else
   printf("Page check failed! Must do RUNSTAT.\n");

// Space check. Calculate the ratio TABSIZE/FPAGES. 70% of total
// space reserved for the table should contain data; therefore, this
// ratio must be greater than ".7".

// SQL returns -1 if RUNSTATS not executed. If RUNSTATS has been
// executed, calculate ratio else print failed msg.

if (fpages != -1) && (avgcollen != -1)
   // calculate ratio
   space_ratio = ((avgcollen+8)*card)/((fpages-1)*4020);
else
   printf("Space check failed! Must do RUNSTAT.\n");

// Overflow check. Calculate the ratio OVERFLOW_ROWS/TOTAL_ROWS. The
// number of overflow rows compared to the number of total rows in
// table should not exceed 5% therefore, this ratio must be greater
// than ".5".

// SQL returns -1 if RUNSTATS not executed. If RUNSTATS has been
// executed, calculate ratio else print failed msg.

if (overflow != -1) && (card != -1)
   overflow_ratio = overflow/card;           // calculate ratio
else
   printf("Overflow check failed! Must do RUNSTAT.\n");
```

Figure 5. C Program to Process Table Statistics (Continued)

```
// Print report

   printf("NPAGES/FPAGES = %d   "page_ratio); // display page_ratio.
   if (page_ratio < .80)                     // If ratio < .8 (80%),
     printf("-Page check failed!\n");        // print failed msg.
     else printf("+Page check OK!!!\n");     // else print OK msg.

   printf("TABSIZE/FPAGES = %d\n"space_ratio);// display overflow_ratio
   if (space_ratio < .70)                      // If ratio < .7 (70%),
     printf("   -Space check failed!\n");      // print failed msg.
     else printf("   +Space check OK!!!\n");   // else print OK msg.

   printf("OFLOW/ROWS = %d\n"overflow_ratio);// display overflow_ratio
   if (overflow_ratio > .05)                   // If ratio > .05 (5%),
     printf("   -Overflow check failed!\n");   // print failed msg.
     else printf("   +Overflow check OK!!!\n"); // else print OK msg.
```

Figure 5. C Program to Process Table Statistics

Index Reorg Check

Three ways to measure index efficiencies are:

- Cluster ratio
- Space usage
- Index levels usage

The cluster ratio is one of the most important index statistics. If this ratio is very low, the DBM will not use the index to perform searches, updates, or deletes. This ratio is calculated by the optimizer using a complex formula. The DBM uses the information stored in the system tables and the DBM configuration file to determine the cluster ratio (CLUSTERRATIO). The CLUSTERRATIO should be .8 or greater. If more than one index is defined for a table, some of the indexes will have low CLUSTERRATIO.

Just as in the table statistics, a measure of the space utilized by the index should be made. The ratio of index size to allocated space is called space usage ratio. The index size is the number of bytes needed for all index entries. The formula determines the size of the unique keys and than adds in the size of the entries for duplicate keys. The size of this entry is four bytes. At least 50% of the space reserved for an index should be in use, so this ratio should be greater than .5. Perform this test when NLEAF is greater than one.

The level usage test verifies that the number of index entries is greater than 90% of the maximum achievable index entries. The ratio is shown in Figure 4b as 90% Max # entries/Actual # entries. This ratio should be less than one. For example, if the maximum achievable number of index entries is 100, 90% of that figure is 90. The ratio would be 1.2, and therefore undesirable, if the index contained only 80 actual entries. Perform this test when NLEVELS is greater than 1.

Test Data Validity

Many developers create test tables and fill them with test data to debug and/or tune their applications. It is important to use data that is unique when conducting performance tests. A way to measure the uniqueness of your data is

to compute the ratio of FIRSTKEYCARD to CARD. FIRSTKEYCARD is found in SYSINDEXES and CARD is located in SYSTABLES. First, define an index on the column(s) of data under test. Then execute RUNSTATS to load the system tables with information on the index. Now you can query the system tables for the information you need. The SQL call you could make is:

```
   EXEC SQL
     SELECT FIRSTKEYCARD,CARD into
:firstkeycard, :card
     FROM SYSIBM.SYSINDEXES A,
SYSIBM.SYSTABLES B
     WHERE A.NAME = 'YOUR_INDEX_NAME'
AND B.NAME = 'YOUR_TABLE_NAME';
```

A value of 1 for this ratio indicates that each key of the table is unique. Values less than 1 indicate duplicate data. This ratio should equal the estimated ratio of the "real life" data. For example, if 25% of the values in real life data will contain duplicate data, the ratio should be (1 - .25)=0.75.

Plan Tuning

The isolation level and record blocking options of a plan have an effect on performance. It is a good idea to check both of these parameters to verify that your plans are bound correctly. The optimal value of these fields is application-dependent. The default is cursor stability and unambiguous record blocking. Figure 6 shows a portion of the "C" code that displays information about each plan in the database.

Authorization Check

Many application programs allow the user to build queries. Sometimes it is desirable to check the privileges of users before allowing them to build queries or even open a database. If you know the userid, you can check the SELECT authority of the user with an SQL call such as:

```
  EXEC SQL
    SELECT SELECTAUTH into
:selectauth
    FROM SYSIBM.SYSTABAUTH
    WHERE TTNAME =
'YOUR_TABLE_NAME' and GRANTEE =
'USERID
```

```
 // Plan Tuning

// Retrieve ISOLATION and BLOCK info for all user tables from SYSPLAN.

// Declare cursor
// "NULLID" is qualifier for system tables used by DBM and QM

EXEC SQL DECLARE C1 CURSOR FOR
  SELECT NAME,ISOLATION, BLOCK
     FROM SYSIBM.SYSPLAN
     WHERE CREATOR <> 'NULLID  ';

// OPEN CURSOR
EXEC SQL OPEN C1;

// While no SQL error, retrieve all data and display to user

while (SQLCODE==0) {
 EXEC SQL FETCH C1 INTO :name, :isolation, :block;
 printf("PLAN %s...Isolation = %c and Blocking = %c\n",name,isolation,block);
}
```

Figure 6. C Program to Collect Isolation and Blocking Levels

SUMMARY

Creating and maintaining a database efficiently requires knowledge of how information is stored in the database. This article explains how DBM stores data and indexes in binary files, and describes the system catalogs that DBM maintains, whose data can be used to analyze database application performance. The data can be retrieved easily using standard SQL, and can be used to tune performance, determine user privilege status, or assess whether a reorganization is needed. Also be aware that a new function, REORGCHK, is available in OS/2 ES 1.0 which automates the process of maintaining a database.

Karen Tyger, *IBM Advanced Workstations Division, 11400 Burnet Road, Austin Texas 78758. Ms. Tyger joined IBM in 1987 and is a senior associate design engineer in the next-generation VLSI graphics group. She was lead programmer for the DBM support team in Austin. She joined IBM in 1987. Ms. Tyger holds a BS in Electrical Engineering from University of Texas. Currently, she is pursuing a Masters in Business Administration from St. Edwards University.*

REFERENCES

IBM OS/2 EE DBM Programming Reference Vol 1 (S01F-0269)

IBM OS/2 EE DBM SQL Reference (S01F-0293).

REORGCHK package. Available on MKTTOOLS and in OS/2 ES 1.0

ACKNOWLEDGEMENTS

The author wishes to thank Tim Malkemus and Anton Versteeg for their contributions to this article. Mr. Malkemus supplied the formulas for the application program section and provided technical expertise. Mr. Versteeg supplied an explanation of the formulas. He also is the author of the REORGCHK program referenced above. It is available in Extended Services for OS/2 1.0 via the Command Line Interface.

International

OS/2: The International Scene

Vicky Gallop

by Vicky Gallop

EMEA DEVELOPER ASSISTANCE PROGRAM

The International Developers Assistance Program (DAP) has been successfully launched in Europe, the Middle East and Africa (EMEA) and was enthusiastically received everywhere. Accepted members for the pilot totalled 140 and correspondence through the online network was plentiful. The pilot project ended 31 December 1991; it was declared a huge success and the full EMEA DAP opened on 13 January. This is expected to grow to 3,000 members.

TO ORDER IBM PERSONAL SYSTEMS DEVELOPER

International subscribers may now order this magazine directly from the publisher. The cost is U.S. $39.95 for one year (four issues), plus a premium for mailing (U.S. $26.00-airmail, $14.00-surface) and a bank charge ($15.00). Call +1 203-366-4000 or FAX to +1 203-366-9100. It may also be obtained through IBM's Systems Library Subscription Service (SLSS), order number G362-0001.

The OS/2 Application Solutions directory is also available for subscription outside of the U.S. - U.S. $19.95 plus mailing ($23.00-airmail, $6.75- surface) plus a $15.00 bank charge. It may also be ordered through SLSS order number G362-0002.

IBM OS/2 COUNTRY CONTACTS

This list of OS/2 contacts by country will be updated periodically as the information changes. If there is no OS/2 contact listed for your country please call your IBM representative or the IBM OS/2 launch manager in your country. If neither can help you, please contact me.

COUNTRY	CONTACT	TELEPHONE	LOCATION
Asia Pacific Group			
Australia	Mel Steiner	Tele: 61-1-634-8991	Sydney
Japan	W. Kumada	Tele: 81-3-3808-4290	Tokyo
Americas			
Brazil	Wagner Okutani de Almeida	Tele: 55-192-65-7138	Sumare
Canada	Ed Rimas	Tele: 1-416-443-4070	Toronto
Mexico	Juan Carlos Fernandez	Tele: 52-5-557-8588 x1846	Mexico
Europe, Middle East and Africa Corporation (EMEA)			
Austria	Hans Dufek	Tele: 43-222-21145 x2731	Vienna
Belgium	Patrick Bulckaan	Tele: 32-2-214-2386	Brussels
Denmark	Poul Hansnes	Tele: 45-45-93-4545	Copenhagen
Egypt	Shamel Abaza	Tele: 20-2-349-2533 x723	Cairo
Finland	Ilkka Ayravainen	Tele: 358-0-4594007	Helsinki
France	Jean-Paul Rambaud	Tele: 33-1-49057390	Paris
Germany	Hans-Michael Obst	Tele: 49-711-785-2417	Stuttgart
Greece	Chryssina Hadjitheodorou	Tele: 30-1-32-219769 x706	Athens
Gulf Countries	Robert Kikano	Tele: 973-210880	Bahrain
India	Tang Wek Soon (covered by Singapore)	Tele: 65-3201066	Singapore
Ireland	Noel O'Rorke	Tele: 353-1-603744 x4361	Dublin
Israel	Arnan Dar	Tele: 972-3-618-700	Tel Aviv
Italy	Virgilio Ferrari	Tele: 39-2-7548-2957	Milan
Korea	Oh Chang Bae	Tele: Korea 519-1457	Seoul
Netherlands	Frans Bik	Tele: 31-20-565-3209	Amsterdam
Norway	Paul Oyan	Tele: 47-2-99-93-83	Oslo
Pakistan	Inayat Ali Ebrahim	Tele: 92-21-525181-190	Karachi

COUNTRY	CONTACT	TELEPHONE	LOCATION
Europe, Middle East and Africa Corporation (EMEA)			
Portugal	Maria Augusta Morgado	Tele: 351-1-542377 x3228	Lisbon
Spain	Jose Maria Castro	Tele: 34-1-397-9502	Madrid
Sweden	Anna Rodholm	Tele: 46-8-793-1000 x1403	Stockholm
Switzerland	Joerg Henseleit	Tele: 41-1-207-2725	Zurich
Turkey	Aysin Berkman	Tele: 90-1-180-0900	Istanbul
United Kingdom	Pat James	Tele: 44-256-475050 x8198	Basingstoke
Yugoslavia	Drago Kodele	Tele: 38-61-325461	Ljubljana
4 gm ROECE*	Carolyn Sandner	Tele: 43-222-21145 x6842	Vienna

*Eastern European countries not including the USSR. In these countries, OS/2 is available without national language support at this time.

Vicky Gallop, *IBM UK Ltd., Mountbatten House, Basing View, Basingstoke, Hampshire, RG21 1EJ, England. Ms. Gallop is a marketing specialist in the OS/2 Technical & Marketing Support Group of the EMEA Personal Systems Centre. Prior to IBM Ms. Gallop worked as a statistician, then made a career move into the computing industry, working for a variety of computer consultancies before joining IBM. Ms. Gallop holds a B.Soc.Sc degree in Economics from the University of Birmingham.*

Systems Application Architecture

CUA '91: What's New?

by Lorraine B. Elder

Two IBM publications describe the details of the Common User Access™ (CUA™) user interface. The Systems Application Architecture™ Common User Access Guide to User Interface Design (order number SC34-4289, known informally as the CUA Guide) and the Systems Application Architecture Advanced Interface Design Reference (SC34-4290, known informally as the CUA Reference) were published in October 1991. These books expand on information in the SAA™ CUA Advanced Interface Design Guide (SC26-4582), published in 1989.

This article discusses the new books and describes some of the changes in the CUA architecture since 1989.

JUST WHAT IS A CUA USER INTERFACE?

A CUA user interface is a graphical user interface based on established principles of user-interface design, field experience, and usability testing. Because a CUA user interface promotes consistency across products, users can transfer knowledge about one CUA product to other CUA products. Some potential benefits of this knowledge transfer include increased user productivity and satisfaction, and reduced user errors.

The CUA user interface is one of the components of the Systems Application Architecture (SAA) platform. Along with the Common Programming Interface (CPI) and Common Communications Support (CCS), the CUA user interface enables developers to produce products that are consistent across SAA operating environments. A CUA user interface can be applied to products for other operating environments as well, such as AIX®.

WHY TWO BOOKS?

Because designers and developers often need different kinds of information at different levels of detail, the guidelines for the CUA user interface are described in two books, each aimed at different (although sometimes overlapping) audiences. The CUA Guide is intended primarily for interface designers, and much of the information in the book can be applied to any kind of graphical user interface, not just a CUA user interface. The CUA Reference is intended primarily for product developers.

To gain an understanding of the concepts behind good user interface design, designers and developers can both read the CUA Guide. When it comes time to write the code for a software product, developers will want to have the CUA Reference at hand.

The CUA Guide

The CUA Guide contains the general conceptual information that interface designers need to know before designing a product with a CUA user interface. The book discusses users' conceptual models of tasks and of human-computer interaction, and it describes design principles that designers can employ to bridge the gap between how users view a product and how programmers or developers view a product. The CUA Guide also describes the key components of the CUA architecture, including basic objects and

Lorraine B. Elder

controls, along with some of the techniques for interacting with them. The final chapter provides an example of one way to develop a product with a CUA user interface. The chapter describes the development process that a group of designers followed to produce a sample software product for use by salespersons in a car dealership. Readers can use the process as a model for developing their own products, or they can adapt the process to their particular situations.

The appendixes offer brief, general guidance on using color and on applying CUA design principles to products that accept touch input or make use of multimedia information. The bibliography lists additional sources of information about software design, user interface technology and techniques, user-centered design, and object-oriented design and programming.

The CUA Reference

The CUA Reference contains a brief overview of the key components of the CUA user interface. Following the overview are detailed implementation guidelines that a developer needs to follow when developing products with a CUA interface. The guidelines fall into two categories: fundamental and recommended. To produce a product that conforms to the CUA architecture, a developer must follow the fundamental guidelines. To make a product even easier to use, a developer should also follow the recommended guidelines when possible.

The guidelines are grouped by topic, such as menus, selection, and views, and the topics are presented in alphabetical order. Each topic includes a definition, an illustration (when appropriate), the fundamental and recommended guidelines, and cross-references to related topics. The fundamental and recommended guidelines are arranged so that developers can use them as checklists for compliance.

The appendixes include a summary of the changes in the CUA architecture since 1989, tips for documenting products with a CUA user interface, and national language support information.

WHAT'S NEW IN 1991?

Major and minor changes appear throughout the new CUA books. Some of the more significant changes are described in the following sections.

Terminology Changes

Several terms have been changed to make CUA terminology more consistent with the terminology used in other user interfaces, and to reflect common usage. In the 1989 definition of the CUA architecture, sometimes two terms were used for the same concept or component. Programmers often used one (usually more technical) term, while documentation for users often contained a different, less technical term. For the 1991 definition of the CUA architecture, terms were consolidated whenever possible, so that developers and users can refer to concepts and components by common terms. Table 1 lists the terminology changes for terms used by users and programmers alike. Additional information about CUA terminology appears in Appendixes A and D of the CUA Reference. *(See Table 1)*

Object Orientation vs. Application Orientation

The 1989 definition of the CUA architecture distinguished the graphical model from the workplace model for applications. The 1991 definition makes no such distinction. Instead, object-oriented products are distinguished from application-oriented products.

An application-oriented product is one in which a user must start an application to work on the files associated with that application. For example, a user starts a word processing application to work with a document file.

An object-oriented product provides a collection of objects that a user can use to perform a task. To work with an object, a user simply opens it or directly manipulates it, for example, by dragging it. The underlying "application" that dictates how a user can interact with an object, as well as how the object can interact with other objects, is completely transparent to a user. For example, to work with a document, a user simply opens

1989 Term	1991 Term
About	Product information
accelerator	shortcut key
action bar	menu bar
action bar pull-down	pull-down menu
cascading pull-down	cascaded menu
dialog box (user term was pop-up window)	secondary window (no longer a user term)
Extended help	General help
function key	shortcut key
Help for help	Using help
hourglass pointer	wait pointer (no longer a user term)
message box	secondary window (used by programmers only; users don't need to distinguish between primary and secondary windows)
pop-up window	secondary window (used by programmers only; users don't need to distinguish between primary and secondary windows)
pull-down	pull-down menu
pushbutton	push button
slider box	scroll box

Table 1. CUA Terminology Changes

a document object, or perhaps drags the document object to a printer object to print the document. The user does not have to start a separate application; indeed, the user does not have to be aware of the underlying processes at all.

Object-oriented products are based on three basic types of objects: container objects, data objects, and device objects. Developers can subclass these basic object classes to create objects appropriate for the tasks of a specific product's users.

Although the long-term direction for the CUA user interface is toward complete object orientation, for migration purposes, the 1991 definition of the architecture still provides for application-oriented products. Generally speaking, products developed for OS/2® 1.3 (or earlier versions) using Presentation Manager® should conform to the application-oriented guidelines, while products developed for later versions of OS/2 or for other operating systems should follow the object-oriented guidelines.

Changes on the Menu Bar

Two of the more noticeable changes in the architecture appear on a product's menu bar. The naming conventions for the first

choice on a menu bar differ according to whether the product is object oriented or application oriented. Likewise, the second choice on a menu bar can differ between object-oriented and application-oriented products. For application-oriented products, the basic choices on the menu bar are File, Edit, View, Help. For object-oriented products, the basic choices are <Object_Class> (the actual name of the choice varies according to the type of object displayed), Selected, Edit, View, Help. Both kinds of products can have additional choices on the menu bar. Figure 1 shows the title bar and menu bar of an application-oriented window. Figure 2 shows the title bar and menu bar of an object-oriented window.

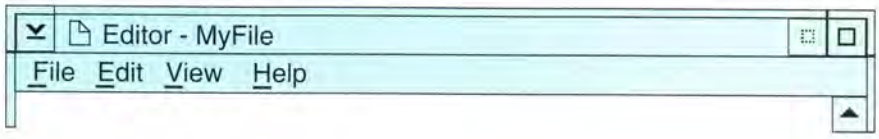


Figure 1. An Application-Oriented Title Bar and Menu Bar

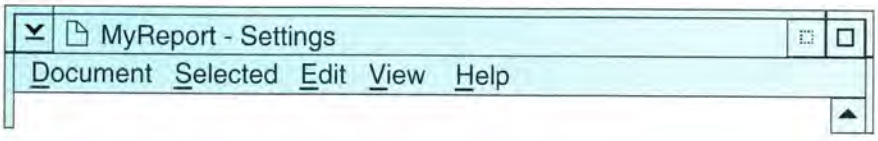


Figure 2. An Object-Oriented Title Bar and Menu Bar

For object-oriented products, the first choice on a menu bar is the name of the object class to which the object represented in the window belongs. For example, if a product includes container objects known as folders, any window opened from a folder displays Folder as the first choice on the menu bar. If a product includes printers as device objects, the first choice on the menu bar for a printer window is Printer. For application-oriented products, the first choice on a menu bar continues to be File, as it was in the 1989 CUA architecture.

The Selected choice in object-oriented products typically appears on the menu bars of container objects and data objects. The pull-down menu associated with the Selected choice contains choices that can be applied to any objects currently selected in the window.

Changes in Windows

Although the basic components of a window haven't changed much since 1989, the content of a window is now described in terms of "views." A view is a way of looking at an object's information. Different kinds of views display information differently, and the type of view affects the name of a window as well as the ways in which a user can interact with the displayed information.

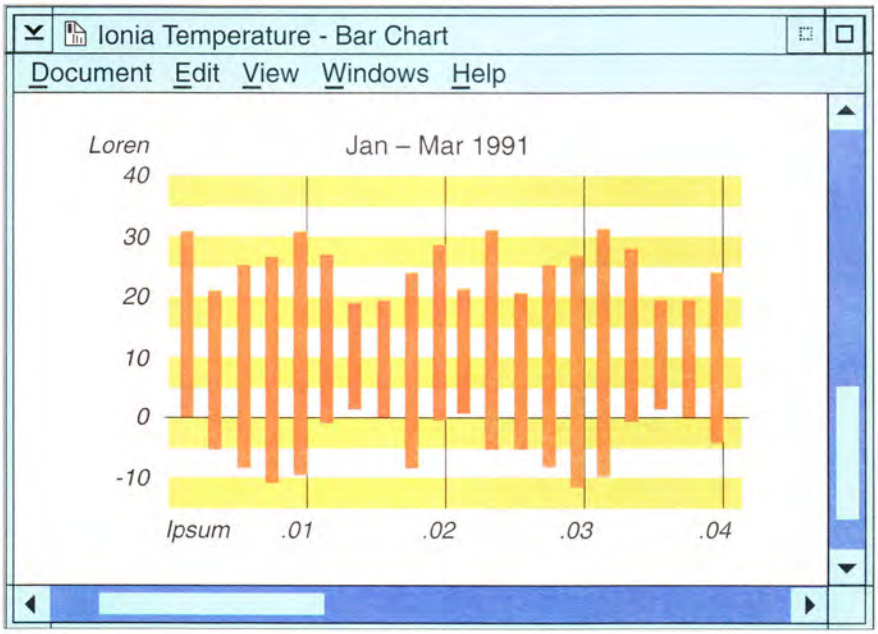


Figure 3. A Composed View

Views

The CUA architecture describes four basic types of views:

- Composed views display the components of an object in relative order.
- Contents views display a list, either ordered or unordered, of an object's contents.

The CUA architecture also describes two specific types of contents views: icons views (graphical representations of an object's contents) and details views (graphical and textual information arranged in columns and rows).

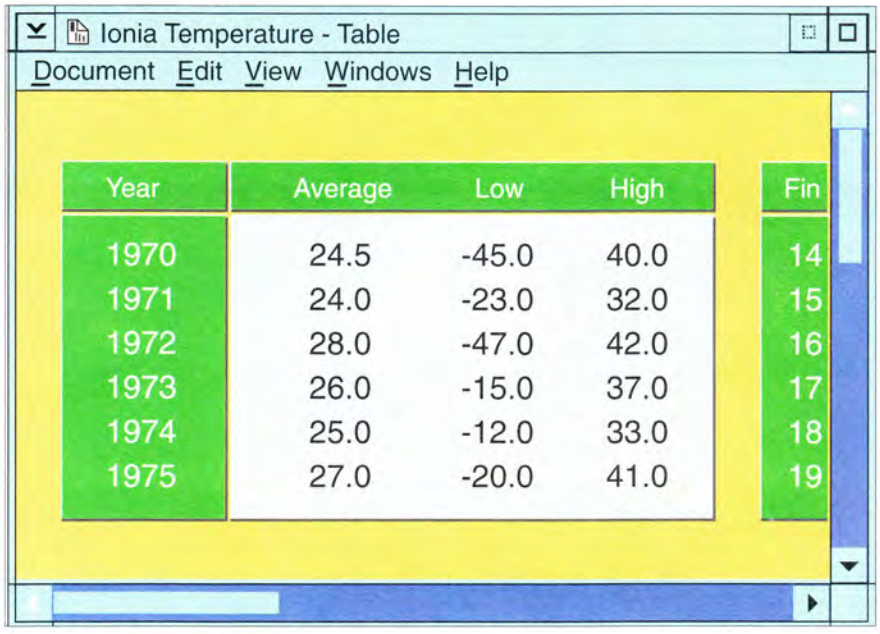


Figure 4. A Contents View

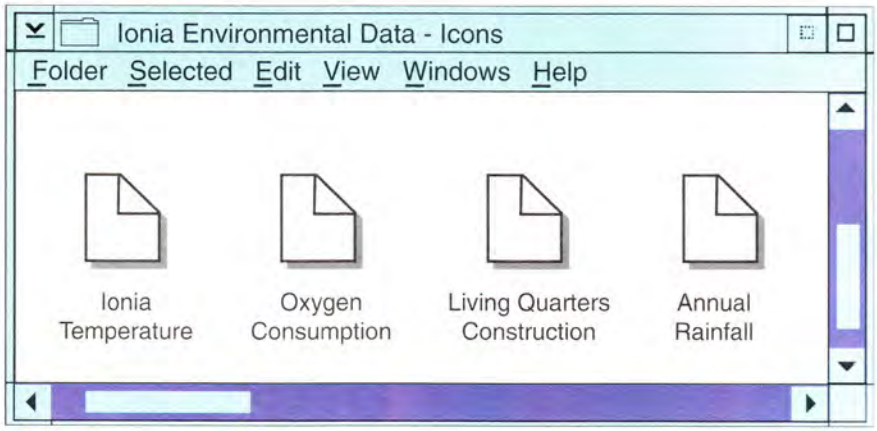


Figure 5. An Icons View

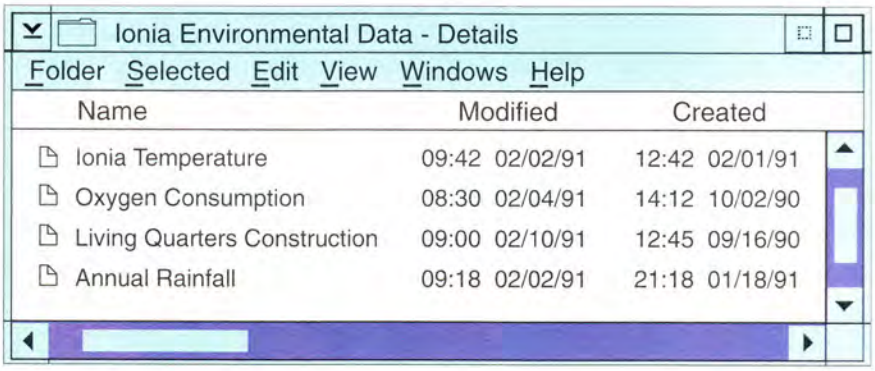


Figure 6. A Details View

- Settings views display information about an object's properties or characteristics. Settings views are often displayed in a notebook control.
- Help views display information that can assist a user in working with an object.

Designers and developers can extend the basic views to meet the needs of a product's users.

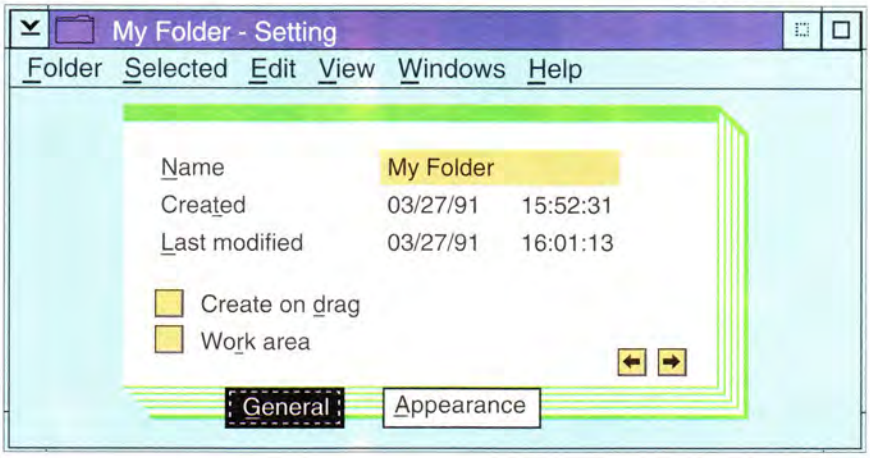


Figure 7. A Settings View

Window-Naming Conventions

In object-oriented products, the name that appears on a window's title bar consists of the name of the object represented in the window and the name of the current view of that object. For example, if a document object is named MyReport and a user opens the document to display a settings view, the window that appears is titled MyReport - Settings.

In application-oriented products, the window title consists of the application name and the file name. For example, if a user opens a file called MyFile with a text editor application called Editor, the window that appears is titled Editor - MyFile. Figures 1 and 2 show the differences in window names.

Secondary Windows, Dialog Boxes, and Message Boxes

The 1989 CUA architecture distinguished secondary windows from dialog boxes, and it included message boxes, which were a special kind of dialog box. Secondary windows could be moved and sized, while dialog boxes, including message boxes, could be moved but not sized. Dialog boxes could be modal or modeless. If a dialog box was modal, a user could not interact with any other part of an application without first performing whatever action was necessary to remove the dialog box.

The 1991 CUA architecture expands the definition of secondary windows to include all windows that are dependent on a primary window, regardless of whether they can be moved or sized. Further, the guidelines recommend that all windows be modeless except when a user must supply additional information before information processing can continue.

Changes on the Workplace

In the 1991 CUA architecture, the workplace is a container object that fills the entire screen. It contains all other objects, and it serves as an area in which users can work with objects to complete tasks. In object-oriented products, each object can be represented on the workplace by one or more icons.

Usually, objects and icons have a one-to-one relationship. However, the 1991 CUA architecture provides a way for a user to create reflections of an object if the user wants to have access to an object from more than one location. Reflections serve as pointers to the original object. Their icons can appear anywhere that the original object's icon can appear. When a user changes an object through a reflection, the change appears in the original object and in all other reflections of the object. However, when a user deletes a reflection, the original object and all other reflections remain intact in their respective locations.

A container object known as the delete folder provides a way for users to delete objects by direct manipulation. To remove an object, a user drags the object to the delete folder. All objects in the folder are deleted at a specific time or interval, which can be set by the user. Until that time, a user can retrieve objects from the delete folder.

Depending on how a product is implemented, users now have a choice of minimizing a window or hiding a window to reduce clutter on the workplace. When a user minimizes an object's window, a graphic of a minimized window appears on the workplace. When a user hides an object's window, the window disappears entirely. In either case, a user can gain access to any open window through a window list.

In the 1989 CUA architecture, the workplace was referred to as the screen background, and only applications that were running could be represented on it, either in windows or as application icons. When a user minimized the window of a running application, the window was represented on the workplace by a minimized-window visual. A user could use a Switch to... action to display a list of all running applications, but not a list of open files. A user could copy files, but the copies were not related in the same way that reflections are related; that is, a user could not change all copies by changing one copy.

WHAT'S NEXT FOR THE CUA INTERFACE?

A brochure, demonstration program, and videotape titled "The CUA Vision —Bringing the Future into Focus" (G242-0215-00, GV26-1003-00) are now available. These materials describe future enhancements to the CUA interface.

SUMMARY

The new CUA books describe the CUA architecture in formats that are convenient for both product designers and developers. The CUA Guide contains conceptual information, and the CUA Reference contains implementation information.

The CUA architecture has changed since 1989. The 1991 CUA architecture distinguishes object-oriented products from application-oriented products, and this distinction is reflected in many components of the CUA user interface, including windows, menus, and icons. The goal of the refined architecture is to improve the usability of products with a CUA user interface. A complete summary of the changes in the architecture appears in Appendix A of the CUA Reference.

ACKNOWLEDGEMENTS

Thanks to Lee Harold, Sue Henshaw, and Juanita Couch for reviewing and contributing to this article.

REFERENCES

The CUA Vision—Bringing the Future into Focus (G242-0215-00)

Systems Application Architecture Common User Access Advanced Interface Design Reference (SC34-4290)

Systems Application Architecture Common User Access Advanced Interface Design Guide (SC26-4582)

Systems Application Architecture Common User Access Guide to User Interface Design (SC34-4289)

Lorraine B. Elder, *IBM Corporation, 11000 Regency Parkway, Cary, NC 27512, (919) 469-6000. Ms. Elder, an associate information developer, joined IBM in 1990. She writes about the CUA user interface, and follows developments in the fields of visual communication, information design, and publication production. Ms. Elder holds a BA in public communications from the University of Alaska, Anchorage, and an MS in technical communication from Rensselaer Polytechnic Institute.*

Application Enablers

The Design of Distributed Applications Using FBSS

by Jesus Ruiz

Financial Branch Systems Services (FBSS) is an IBM™ product that enhances several operating systems (DOS, OS/2™, OS/400™ and in the near future AIX™) with the system services needed to design and implement truly distributed applications using the client-server mechanism. In addition to these basic distributed system services, FBSS also provides additional functions that help manage and control the complex environments found in practical implementations. These functions normally take the form of already made servers which are shipped with the product. The customization program provides for the specification of the actual LAN topology at runtime and the definition of the characteristics of each server in the LAN to fit with their expected usage. Additionally, several financial servers are provided to help application development in this area. But one of the strengths of the product is the ability of users to add to this already comprehensive distributed system platform their own user-written servers, as part of the effort in writing 'mission critical' distributed applications.

THE ROLE OF USER SERVERS IN DISTRIBUTED APPLICATIONS

Consider the following requirement: you have a PS/2™ and one device you want to attach to it. Taking an example from the banking environment, the device could be a Financial Passbook printer. This is a special printer which lets us print items on a passbook (as its name implies). The printer is special in the sense that it doesn't behave like a 'normal' printer. You have to define the format of the passbook, and the feeding of data to the printer is done in a special way. Thus you can not just connect the printer to the printer port and print data in the same way as you would do with an IBM Proprinter™.

However, when you purchase the printer, you also get a device driver for it and several utility routines which allow you to use the printer from a high level language. The routines themselves are not very 'high-level' in functionality, but at least they relieve you from the burden of taking care of many low-level details when printing.

Now you want to write an application that, as a result of its processing, writes to the passbook printer. Knowing that this may not be the last application you have to design using this printer, you would probably structure the application in very well defined modules, using structured design techniques.

One of the modules 'encapsulates' the printer and hides the details of it to other modules of the application which want to use the printer. This module manages internal data (printer status and so on) and 'exports' routines which are simple to use, such as 'Define THIS passbook format,' 'Print THIS data in THAT format,' and more. So the module can be easily reused in case you need to write another application. Also, if you change the printer to another model (or another vendor), chances are that the change only implies rewriting the module, not the whole application.

You then write the rest of the application modules, which can access the printer just by

calling up the routines exported by the printer module.

With the application you just developed, you need a passbook printer attached to every PS/2. But the printer is expensive, and it seems that the normal usage of the printer is low, so it would be convenient to have the printer shared among several users of your application in the LAN. You find that if there is a printer for each pair of users in the LAN, normal operation of them is not affected while the cost of the total system is lowered considerably. Sharing the printer among more than two users presents some limitations, because the probability of contention for the printer is high. Also, the user has to manually feed the passbook into the printer so the printer must be physically close to the user. This is easy to achieve with two users (you put the printer mid-way between them so they reach it without moving from their seats), but not so easy with three or more.

So you decide to attach one printer to a pair of PS/2s that are physically close to each other. You can then build a LAN of any size by just grouping several such pairs of configurations. If the LAN happens to have an odd number of PS/2 there would be one with a printer for its exclusive use.

The problem now is how to share the printer. Remember that the printer is a special one, so you can not use the IBM PC LAN program (if the PS/2 runs DOS) or the IBM LAN Server (if it runs OS/2). It seems that the only solution is to write a server specifically for this printer, shared by multiple requestors. This way, your LAN has the 'standard' servers for file and data sharing, plus your new server.

FBSS is an IBM product that allows the user to perform this type of task easily. It enhances the operating system with distributed system services to allow the easy implementation of client-server computing environments.

THE PATH FROM A NORMAL APPLICATION TO A DISTRIBUTED ONE

The following paragraphs introduce the concept of a distributed application in a LAN, and how we can get to it in a simple way from a 'normal' monolithic one designed in the conventional way. The description will be a little bit over-simplified, but it will describe the important aspects of the process.

Let us assume that the normal approach when designing a standalone application (one that will run on a single machine) is to structure the application in modules, related hierarchically among them.

Data and other resources are encapsulated by routines providing access via an interface which isolates the modules using the resource from the implementation details. External modules should manipulate the resource via the routines exported by the module which is the 'owner' of the resource.

A given module provides certain services for other modules to use. Following the hierarchical structure and in order to provide these services, the module may need to access the services provided by other sub-modules.

We can see that there is a client-server relationship among application modules depending on the hierarchy. Some of the characteristics of this client-server relationship are:

- We can call client module to a module using the services provided by another module. The provider of the services can be called a server module.
- Multiple client modules can use the services provided by the same server module. In a single-threaded application, the client modules use the server module in a serial fashion (one after the other). In a multi-threaded application (as can be done in OS/2), the designer may decide to allow concurrent use of the server module.
- A server module can be at the same time client of other modules. We can say that it uses the other modules and encapsulates them to provide higher-level services to its clients.

- The normal way of accessing a server module is by means of what we call a 'Local Procedure Call'. The client calls a routine and control is then transferred to the server module. The term 'local' refers to the fact that both client and server modules reside in the same application code. This is the normal calling convention in most high-level programming languages such as Pascal, C, or Cobol.

- The 'conversation' between a client and a server follows the call-return model. When a client wants a server to perform some operation, it calls the specific routine in the server which handles the task. The server then performs the processing and returns to the client, passing back any results of the processing. This mechanism is inherently synchronous, once the client has called the server, it does not receive control back until the server completes processing and returns to the client.

- We see that this 'flavor' of client-server terminology applies very well to the standard and well-established techniques of structured application design.

- Some of the modules which compose the application may be implemented by the application developer. Others may be supplied by third-party vendors. The modules are used by the application developer as building blocks, and the only thing that should be known about the modules is their functionality and their external interfaces.

It would be nice if there were an easy way to convert these client and server modules into 'real' clients and servers. This means to get one of the server modules and transform it (as slightly as possible) to a standalone entity: the 'real' server. 'Standalone' entity means that the server is then a program separate from the rest of the application. This program should be able to load on a machine in the LAN and respond to requests coming from 'real' clients. The client modules should also be transformed so they request services from the 'real' servers instead of from the server modules.

If we apply these transformations to selected modules of the 'monolithic' application, we then have an application which is composed of several separate entities: 'real' clients which request services from 'real' servers, which at the same time may request services from other 'real' servers. The hierarchical structure of the application is still the same.

In the example of the special printer, our problem would be solved if we can convert the printer module into a server which can be accessed from several clients.

If the operating system supports a mechanism by which requests from a client in one machine can reach the corresponding server in any other machine of the LAN, without the requester knowing the physical location of the server, then we can have a truly distributed application over the LAN. The only other requirement for the operating system support is that servers can be loaded in **any** of the machines in the LAN, so we can distribute the application in the most appropriate way.

This approach to client-server computing generalizes the one that is usually presented in current literature, which tends to present the server portion of an application as the heart of the application, usually running on only one machine. In this view, the client only focuses on user-interface issues. The logic of the application is totally concentrated. But our aim in this article is to provide the mechanisms to be able to distribute not only the data managed by the application (as can be done using the IBM Database Manager™, for example), but also the logic of the application. And to be able to decide the actual physical distribution of the application logic after the application is designed and implemented, not before.

This is only possible if users are allowed to write their own servers so they can design their application as being composed of a hierarchical structure of servers. In this approach, the servers replace the traditional 'application modules', but the logic of the application remains unchanged. Here is where the concept of 'user server' appears. Writing a user server should not be much

more painful than writing a 'standard' module with the same functionality, for this new approach to be useful. Of course 'standard' servers can be integrated in this new distributed application structure, if the designer needs support for standard printing services, communication services or database services, for example.

Without pretending to be exhaustive, we can define server designs as two tasks:

1) Implementing the procedures to access and control the resources managed by the server. For a database server, this would be all of the data handling and control procedures. This is why the server was written: to control the resources.

2) Implementing the mechanisms needed to allow requesters to access the server so they are able to use the resources managed by the server. This is what converts a server module into a server process.

Task 1 is specific for each server, differentiating it from the other servers. Database management, printing services and communications services belong to this task.

Task 2, however, is common to every server. The actual way the server implements it may be different, affecting things like access performance of access of server from requesters, type of requesters allowed (OS/2 only, DOS requesters, ...), etc.

Even though this task may seem fairly simple, it is not as simple as accessing the server from clients. To be able to implement 'mission critical' client-server applications, this layer should comply with several requirements similar to those met by the corresponding layer in 'standard' servers like OS/2 Database Manager or the OS/2 Communications Manager™.

The issue of user servers has been addressed only partially up to now in the literature. Communications primitives tend to be confused with an architecture to design and write user servers. Many supposed client-server samples have been given which force both clients and servers to make explicit use of communications verbs to implement very simple client-server applications. Some examples of these communication primitives are NETBios, named pipes and APPC (LU 6.2)™.

As mentioned previously, any existing 'standard' server has implemented this layer on its own, because it relies on an operating system which does not yet support an architecture for writing servers. They may use any or all of the above communication primitives classes internally, but the clients are transparent to what communications protocol is actually used to reach the server. You don't have to explicitly use named pipes or NETBios to make a request to the IBM SQL Database server, for example. This layer not only achieves the necessary connectivity among the machines, but also provides transparency of the location of the server to requesters (something you can not say about named pipes, because you need to know both the name of the server pipe and its machine name). Another limitation of named pipes is that pipes can only be created by processes residing in a machine with the IBM LAN Server installed. If you wish to implement a really distributed application with user servers using named pipes, you will end up with a LAN Server for every machine (the LAN Server, not the LAN Requester)-something which does not seem very acceptable.

So what can we do? Are we alone in this task?

WHAT IS FBSS?

The FBSS program family combines IBM PS/2 and IBM AS/400™ machines with the operating systems (DOS, OS/2 and OS/400) to form a client-server data processing system with a consistent application platform to allow the building of applications distributed over a LAN of these machines. There is a statement of direction on FBSS for the AIX operating system to provide the same functionality on the RS/6000™ and allow it to become a member of the FBSS family.

FBSS is an extension to the standard operating systems. It enhances their functionality providing distributed system services and defining an architecture to allow the design and implementation of truly distributed applications based on the client-server paradigm.

In addition to this and to allow the control and management of complex distributed processing environments, FBSS also provides additional services. Some of them implemented as separate programs, some others as already-made servers built on top of the client-server mechanisms of the distributed processing layer. Some examples are:

- A customization utility to allow the definition of FBSS LANs. User servers are integrated in this utility. The topology of the LAN is defined: number and name of the machines, the servers that will be used and the services which will be available to the requester. Additionally, each server is tailored to the exact function it will have to provide at runtime. As a result of this customization process, the software needed for each machine in the LAN is generated ready for installation.

- The System Manager and the System Manager Operator servers are a key element of total systems management.

- The Remote Change Management System server allows the software to be distributed from a central host to the (possibly) many LANs comprising the entire enterprise. Netview/Distribution Manager™ is used at the host for this purpose.

- Trace servers are used to collect trace data from the distributed components of the LAN to diagnose problems.

In addition to these basic system services, there are some servers which provide more functionality needed to implement distributed applications:

- The SQL Database server allows a requester to access SQL data located in any supported machine (OS/2 and OS/400, in the near future AIX) in a transparent manner. The requester accesses the data in the same way independently of where it is located.

- The Shared File server provides additional database services for indexed and sequential data access. These services are used by other servers provided with FBSS for their internal data handling processes.

- The Communications servers provide SNA and native X25 connectivity from an FBSS LAN to the outside world. As it happens with the SQL Database server, the requester doesn't know the physical location of the Communications server. In DOS, it is a very complex server. In OS/2 and OS/400, it is a layer on top of the communications services already provided by strategic IBM products (the IBM Communications Manager in OS/2, for example). But all of these communications servers are accessed the same way from requesters.

FBSS is evolving into a cross-industry product, but its origins are in the financial world. FBSS was born due to the advanced requirements of the financial industry towards a distributed processing environment. The intention was to down-size traditional host-based applications using the then emerging LAN technologies. Not in vain FBSS stands for Financial Branch System Services (although the basic distributed processing services provided are general purpose in nature). To simplify these tasks, FBSS provides additional servers packaged with the product, comprising:

- The Electronic Journal server.
- The Store for Forwarding and the Forwarder servers.
- Several servers to support the standard financial Input-Output devices (Magnetic Stripe Reader/Encoder device, Personal Identification Number device, financial printers, etc.).

Due to the heavy use of FBSS in the financial world, the FBSS user community has also developed additional servers whenever the need for sharing special devices arose and they were not specifically supported by the standard product. They are not provided with FBSS, but they integrate into the total solution. Requesters access them in the same way as any other in the system. Samples include servers to support data encryption devices, specialized printers, document readers, messaging services, etc.

Some IBM products also provide FBSS servers to facilitate their integration into a LAN. Examples are the IBM Transaction Security System™, the IBM Touch Application Enabler™ and the IBM 4737 Self Service Transaction Station™.

But FBSS servers not only serve to share and control special devices in a LAN. They also facilitate the distribution of the application logic to the processors most suitable for that. Examples of it are countless and depend on the actual application.

FBSS has been in the market since 1987, and nearly 250,000 licenses have been installed worldwide, mainly in financial institutions.

The following paragraphs present by example some of the features of the distributed processing architecture provided by FBSS. Error handling has been deliberately omitted to make the structure of the sample easier to understand.

THE STANDALONE APPLICATION

The code for the sample application is listed in Figure 1.

```
/* Sample of standalone application */
#include <stdio.h>

#define INPUT_SIZE  64
#define RESULT_SIZE 64

char input[INPUT_SIZE];
char output[RESULT_SIZE];

main()
{

   /* Perform any required initialization */
   initialize ();

   /* Get a string from the user */
   printf ("Enter string to transmit:\n");
   gets (input);

   /* Translate the string */
   translate (input, output);

   /* Print the returned string */
```

Figure 1. Example of Standalone Application (Continued)

```
    printf ("Result: %s\n", output);

    /* Terminate the program */
    exit (0);

}

int initialize ()
{

    /* Any necessary initialization would be performed here */
    return (0);

}

int translate (char * input, char * output)
{

    /* Copy the input buffer to the output buffer */
    strcpy (output, input);

    /* Process the buffer translating it to uppercase in place */
    strupr (output);

    /* Return to caller */
    return (0);

}
```

Figure 1. Example of Standalone Application

The function performed by the application is very simple. It prompts the user to enter a string of data, which is then translated to uppercase and displayed on the screen. The steps to perform this are:

- The 'main' procedure of the application first calls the 'initialize' function. Any required initialization code would go here (in this sample, the routine is void).
- Then the user is prompted for the string, which is put in the 'input' field.
- The routine 'translate' is called to perform the process. The results are put in the 'output' buffer.
- The 'output' buffer is displayed on screen.

Our task is now to transform the application so we can distribute its logic. Obviously, the only 'logic' we can distribute in this sample is the translation process. We will then convert the function 'translate' into a server. The main procedure of the application will be converted to a client of this server.

THE CLIENT PART OF THE DISTRIBUTED APPLICATION

Figure 2 shows the complete code for the client part. We see that it is very similar to the standalone application (as a matter of fact, the 'main' function is exactly the same).

```
/* Sample client program */
#include <ehcdefc.h>
#include <stdio.h>

#define INPUT_SIZE  64
#define RESULT_SIZE 64

char input[INPUT_SIZE];
char output[RESULT_SIZE];

#define THE_SERVER       "MYSERVER"
#define FBSS_SUPERVISOR  "SPV      "
#define INIT             0x494E
#define TRANSLATE        1

EHC_CPRB ctrl_blk;       /* The control block used to talk to FBSS
*/

main()
{

   /* Perform any required initialization */
   initialize ();

   /* Get a string from the user */
   printf ("Enter string to transmit:\n");
   gets (input);

   /* Translate the string */
   translate (input, output);

   /* Print the returned string */
   printf ("Result: %s\n", output);

   /* Terminate the program */
   exit (0);

}

int initialize ()
{
   /* Register to the local FBSS Supervisor */
   ctrl_blk.ehcfunct = INIT;
   memcpy (ctrl_blk.ehcserver, FBSS_SUPERVISOR, 8);
   RMTREQ (&ctrl_blk, EHC_RESERVED);
```

Figure 2. Client Part of Distributed Application (Continued)

```
    /* Return to caller */
    return (0);

}

int translate (char * input, char * output)
{
    /* Set up the control block */
    memcpy (ctrl_blk.ehcserver, THE_SERVER, 8);
    ctrl_blk.ehcfunct = TRANSLATE;
    ctrl_blk.ehcqdataad = input;
    ctrl_blk.ehcqdatal = strlen (input) + 1;
    ctrl_blk.ehcrdataad = output;
    ctrl_blk.ehcrdatal = strlen (input) + 1;

    /* Send the string to the server for translation */
    RMTREQ (&ctrl_blk, EHC_RESERVED);

    /* Return to caller */
    return (0);

}
```

Figure 2. Client Part of Distributed Application

Some initialization code has been put into the 'initialize' routine. This step should be performed only once during the entire life of the client, to allow FBSS to know about the client. Once this is done, the client can use any server(s) in the whole LAN as many times as it wishes.

The routine RMTREQ is used for this purpose. It accepts the address of a control block as its first parameter. The contents of this control block determine the actual process performed by FBSS. The most important fields in this control block are the server name, the function name and pointers to input and output data. For now, it is enough to know that by setting the server name to "SPV" and function to INIT, the client is registered and added to the FBSS LAN. The client is now ready to perform the real thing.

The next step in the transformation has been to replace the code in the routine 'translate'. Instead of the translation code (which will be put in the server), we find code to send a request to the server. Together with the request, we send the input data. The server will be responsible to return the translated data.

Again, the routine RMTREQ is used for sending the request. This time, the server name has been set to "MYSERVER". We could have chosen any suitable name. Remember that this name refers to the server process, not to the name of the machine where the server is running. FBSS will take care of the request, routing it to the machine where "MYSERVER" runs. As a given server process may export more than one function, we have to specify the function in which we are interested. In this case, we put in the appropriate field of the control block the only function exported by our server: TRANSLATE.

The function code is an integer with an arbitrary value. The only requirement is that both client and server use the same value to designate functions. In this sample, we have chosen the number "1" as function code for TRANSLATE. If we were to add functionality

to our server, we would define more function codes corresponding to these new functions.

The last step before actually calling RMTREQ is to set the appropriate fields in the control block pointing to the input and output buffers. Once RMTREQ returns, the server has processed the request and the translated data is in the output buffer.

THE SERVER PART OF THE DISTRIBUTED APPLICATION

Shown in Figure 3 is the server part of a program with its own main processing function. The first step is to register with FBSS so the server is known in the LAN and can be used by clients. This is done by means of the SRVINIT call.

```
/* Sample server program */
#include <ehcdefc.h>
#include <stdio.h>

#define TRANSLATE   1

extern short far pascal CALCULATE_SIZE (void);
void process_requests (void);

main()
{
   short size;

   /* Register to the local FBSS Supervisor as a server */
   SRVINIT (CALCULATE_SIZE(), 0, process_requests, EHC_RESERVED);

   /* Process the requests from clients */
   process_requests ();

   /* Terminate the program */
   exit (0);

}

void process_requests (void)
{
   EHC_CPRB ctrl_blk;

   /* Start the loop for serving requests coming from clients */
   for (; ; ) {

      /* Get a request from the server queue */
      GETREQ (&ctrl_blk, EHC_RESERVED);

      /* Check that the request is the TRANSLATE one */
      if (ctrl_blk.ehcfunct == TRANSLATE) {

         /* Translate the string */
         translate (ctrl_blk.ehcqdataad, ctrl_blk.ehcrdataad);

         /* Indicate the length of the buffer we are returning */
```

Figure 3. Server Part of Distributed Application (Continued)

```
        ctrl_blk.ehcreplddlen = ctrl_blk.ehcqdatal;

      } /* endif */

      /* Send the result back to the client */
      RMTRPLY (&ctrl_blk, EHC_RESERVED);

   } /* endfor */

}

int translate (char * input, char * output)
{

   /* Copy the input buffer to the output buffer */
   strcpy (output, input);

   /* Process the buffer translating it to uppercase in place */
   strupr (output);

   /* Return to caller */
   return (0);

}
```

Figure 3. Server Part of Distributed Application

The routine CALCULATE_SIZE, which is called also in this step, calculates the size of the server in case that it runs on DOS. This is only necessary if the server is going to run in a DOS machine. In this case, the server will be converted automatically to a Terminate and Stay Resident (TSR) by FBSS. This step has been put in the sample to show how easy it is to write a server which is able to run on any FBSS supported machine. On DOS, FBSS adds some multitasking features which allow servers to be loaded and run in the background while the foreground application is being used. Of course on OS/2 and OS/400 the native operating system multitasking services are used to run servers in the background.

Then the main procedure calls the 'process_requests' routine which will be responsible for receiving requests from clients, processing them and replying.

The 'process-requests' routine is basically a loop of the following steps:

- Call GETREQ to get a request from the server queue. FBSS automatically manages a queue of requests for every server. As a server may be used by several clients at the same time, the queue is needed to temporarily store incoming requests received while the server is busy processing the current one. If the queue is empty at the time of the call, the server is blocked waiting for the next request.
- The request is parsed and if it corresponds to the TRANSLATE one (remember that a server may implement more than one function), the 'translate' routine is called. This routine is totally identical to the original one in the standalone application.
- Call RMTRPLY to send the results back to the client.

SOME COMMENTS ABOUT THE SAMPLE

Even though the sample is functionally simple, it shows several important characteristics:

- The client doesn't know the physical location of the server. It only knows its name and the set of functions it can perform. In this sample the server "MYSERVER" only exports one function.
- The sample server is portable at the source code level. This means that the source code provided can be used to generate a server which can be loaded in any of the operating systems supported by FBSS. This is a consequence of the server not using any feature specific to a given operating system.
- The sample server automatically provides multiuser support. FBSS takes care of it, by managing a queue of requests from clients.
- Connectivity between clients and servers is managed automatically by FBSS. Both clients and servers have only to worry about telling their local FBSS Supervisor that they are ready to start working.

In the sample the server is used serially by clients. But FBSS provides mechanisms to allow servers to implement multiple transactions for multiple clients simultaneously (a transaction being in this case a series of related calls from a client to a server).

Services are provided for both clients and servers to allow powerful asynchronous parallel processing in the LAN. A client can have several servers in the LAN working simultaneously in different tasks. While the servers are working, the client can perform other tasks or just wait for the completion of any or all of the server tasks.

SUMMARY

We have seen how easy it is to write a user server using FBSS. Writing user servers lets us not only share devices and other resources in a LAN, but also distribute the logic of a sophisticated application in the most convenient way.

FBSS can be purchased from IBM through your local IBM Marketing Representative or IBM Agent.

REFERENCES

IBM FBSS Licensed Programs Family General Information, (GC19-5172).

IBM FBSS Licensed Programs Family Program Description, (SC19-5176).

Jesus Ruiz, *IBM Spain, European Banking Systems Barcelona Development Center. He is a member of the Architecture & Advanced Design group in BDC. He joined IBM in 1986 as a developer of the FBSS client/server mechanism. He has a BS in Telecommunications Engineering from the Politechnic University in Barcelona.*

International Business Machines Corporation
P.O. Box 1328
Boca Raton, FL 33429-1328

Printed in U.S.A.